CS385 Machine Learning Project I Final Report

Quan Luo, F1603021, 516030910506

***Abstract*—This is a report on CS385 Machine Learning Course Project I. In the project, we implement logistic regression, fisher model, SVM and CNN on face classification and try those models on face detection. In addtion, feature visualization using PCA & t-SNE is included, too. Need to say that, since the project is experiment-faced, thus the theoretical part here is understated and I focus more on the experiment part.**

I. Data Preprocessing

AS Required in our project, I download the FDDB dataset and do some preprocessing to it. The dataset gives us the face by a bounding eclipse instead of box. So first as required I turned the given eclipse into a box and then get the HoG feature using Python-OpenCV package. Next, we generate the negative samples according to requirement by generating 9 negative samples from one given picture by resizing the whole image and do padding along the axis of the given face images. They can be visualized as Fig.1.

Fig. 1. The first row, we show the face box, cutted face and its HoG feature for positive samples while second row demonstrates two kinds of negative samples and one of their HoG features

From the first part of data preprocessing, we get the needed dataset stored in pickle. Thus we are able to test all kinds of classifiers on the dataset and get the following process work efficiently.

II. Logistic Regression

A. Theoretical Analysis

Consider a dataset with n training examples, where $X_i^T = [x_{i1}, ..., x_{ip}]$ consists of p predictors, $y_i \in 0, 1$ is the boolean label. We assume that :

1) All dimensions x_{ij} in x_i are conditionally independent given y_i
2) $P(x_{ij}|y_i) \quad Gauss(\mu_j, \sigma_j^2)$
3) $P(y_i|X_i) = Bernoulli(p_i)$, i.e. $P(y_i = 1|X_i) = p_i$, $P(y_i = 0|X_i) = 1 - p_i$

Let p_i denote the probability of $P(y_i = 1|X_i) = p_i$, which is estimated by the regression model. We can assume that $logit(p_i) = log\frac{p_i}{1-p_i} = X_i^T\beta$. Then we know that

$$p_i = sigmoid(X_i^T\beta) = \frac{e^{X_i^T\beta}}{1 + e^{X_i^T\beta}} = \frac{1}{1 + e^{-X_i^T\beta}}$$

We can denote the log-likelihood as $logP(\beta) = \sum_{i=1}^{n}[y_i^* X_i^T\beta - log(1 + expX_i^T\beta)]$ and then we are going to maximize the log-likelihood, which equals to maximize the plausible explanation to the observed data. It also gives us the logisitic loss

$$L(y_i, X_i^T\beta) = -[y_i X_i^T\beta - log(1 + exp(X_i^T\beta))]$$

.

As for optimization method, we try two kinds of optimization algorithm, namely SGD & Langevin dynamics. SGD(Stochastic gradient descent) gives us the idea of mini-batch. Instead of learning by every input samples, we randomly selects a mini-batch and replace loss by the average of this mini-batch and go to the optimization step.

Langevin dynamics is derived from the inception of Brownian motion defined by $X_{t+\delta t} = X_t + \sigma\epsilon_t\sqrt{\delta t}$ where $E(\epsilon_t) = 0$ and $Var(\epsilon_t) = 1$. Then according to the central limit theorem, we can get the optimization step equation

$$X_t = x + \sigma\sqrt{t}\frac{1}{\sqrt{n}}\sum\epsilon_i$$

By the Langevin dynamics we are able to do optimization step as same as SGD according to different optimization equations.

B. Experiment

Next I implement the logistic regression with two distinct kinds of optimization process. The accuracy is followed.

Method	Train Accuracy	Test Accuracy
SGD	0.9312	0.9504
Langevin	0.9273	0.9339
Sklearn SGD	\	0.9675

TABLE I
Logistic Regression Accuracy

Next we show their loss decreases as follow. We can see that they are converged properly with our optimization method and loss decreases as we expected. Fig.2 shows the loss history.

We can see that Langevin dynamics starts better. But finally SGD gives us a better result. That's because technically the Brownian Move makes the model move to the chaotic side, which is great for start but bad for the end.

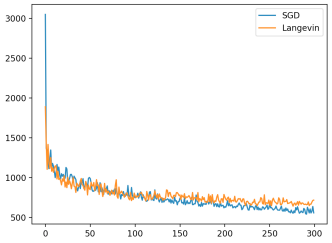

Fig. 2. The loss history of Logistic Regression with SGD & Langevin Dynamics

III. Fisher Model(LDA)

A. Theoretical Analysis

The fisher model, or say LDA(Linear Discriminant Analysis) try to do dimension reduction on given features. Consider that if we choose reducted dimension equals 1, then it can be used by classification. Let us consider two classes $y = +1, -1$ and divide the training set into two parts namely Ω^+ and Ω^-. In the scenario of linear discrimnant analysis we assume that

1) $\forall X_i \in \Omega^+, p(X_i|y=+1) \quad N(\mu^+, \Sigma^+)$
2) $\forall X_i \in \Omega^-, p(X_i|y=-1) \quad N(\mu^-, \Sigma^-)$

Then the task is to learn a linear function that projects X_i to $z = X_i^T\beta$ so that the projection maximizes the inter-class variance and minimizes the intra-class variance. The intra-class & inter-class variance is given as

$$\sigma^2_{between} = [(\mu^+ - \mu^-)^T\beta]^2$$

$$\sigma^2_{within} = n_{pos}\sigma^2_{pos} + n_{neg}\sigma^2_{neg}$$

where $\sigma^2_{pos} = \beta^T\Sigma^+\beta$ and $\sigma^2_{neg} = \beta^T\Sigma^-\beta$. Then we can get the objective $S = \frac{\sigma^2_{between}}{\sigma^2_{within}}$. The problem can be transformed into an optimization problem that $max\beta^T S_B\beta \ s.t\beta^T S_W\beta = 1$ where $S_B = (\mu^+ - \mu^-)(\mu^+ - \mu^-)^T$ and $S_W = n_{pos}\Sigma^+ + n_{neg}\Sigma^-$. And after induction we can get the solvation that

$$\beta \propto S_W^{-1}(\mu^+ - \mu^-)$$

B. Experiment

I implement the fisher model and the accuracy and the two variances are shown below. Need to mention that, in practical problems like our face detection, S_W may be singular and thus doesn't exist the inverse matrix. However we can do a simple transformation that $S_W = S_W + \epsilon I$ where ϵ is small enough. In the code it's also shown the trick.

Method	Intra Var	Inter Var	Acc
LDA	6.04e-08	2.759e-07	0.9674
SKlearn	\	\	0.9706

TABLE II
LDA Results

We can see that our LDA model actually minimize the intra-variance and maximize the inter-variance and get the great result with high accuracy.

IV. SVM

A. Theoretical Analysis

SVM(Support vector machines) use a vision of convex optimization to see the classification task. The related knowledge on Lagrange Multipliers are written in the appendix. The convex optimization problem can be shown as follow

$$min_{w,b}\frac{1}{2}||w||^2$$

$$s.t\forall i, -y_i(w^T X_i + b) + 1 \leq 0$$

The loss function can be constructed by Lagrangian method that $L(w,b,\alpha) = \frac{1}{2}||w||^2 - \alpha_i[y_i(w^T X_i + b) - 1]$ and we are going to $min_{w,b}max_\alpha L(w,b,\alpha)$. However the theory until now is only capable to solve linear questions. Diving into non-linear part, we can use kernel function to add non-linearity. The feature $\Phi(x)$ can be dealed by

$$w^T\Phi(X_i) + b = _{j=1:\alpha_j \geq 0} \alpha_j y_j K(X_j, X_i) + b$$

The kernel functions are denoted mainly by the "similarity" of two features. A typical kernel function RBF Kernel is $K(X_i, X_j) = exp(-\frac{||X_i - X_j||^2}{2\sigma^2})$ can measure the similarity between X_i and X_j.

Next I want to talk about QP & SVM since my homemade SVM is using the view of Quadratic Programming to solve. SVM can be seen as the follow optimization problem

$$min_{b,w}\frac{1}{2}w^T w$$

$$s.t.y_n(w^T x_n + b) \geq 1, \forall n$$

Compared to QP we can get that the parameters in QP problem namely $U = \begin{bmatrix} b \\ w \end{bmatrix}$, $Q = \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix}$, $P = 0$, with constraints $a_n^T = y_n \begin{bmatrix} 1 & x_n^T \end{bmatrix}$, $c_n = 1$. Getting these parameters we can solve SVM as a Quadratic Programming problem with some known packages.

B. Experiment

I implement SVM by myself using cvxopt (a convex optimization package) and directly use SK-learn both.

Method	Kernel	Accuracy
MySVM	RBF	0.8155
MySVM	Linear	0.8506
Sklearn SVM	RBF	0.9861
Sklearn SVM	Linear	0.9271
Sklearn SVM	Poly	0.9890

TABLE III
SVM Accuracy

Not shown in the table, SVM RBF kernel can only run max iteration for 2000 rounds to get the high accuracy while linear and poly kernel have to deal with nearly 10000 rounds to get the listed accuracy. We can see the difference between different kernel and that RBF Kernel can actually measures the similarity between two features.

In addition, we visualize those support vectors as follow. By the way there are totally 3326 pictures of support vectors.

Fig. 3. Some samples of support vectors using RBF Kernel SVM

Thus we only choose a few to visualize as Fig.3. We can see these are some really difficult images to be classified correctly.

Need to explain that, my homemade SVM has bad performance mainly due to 2 reasons. Firstly SK-learn SVM doesn't use the method of convex optimization. I first transform it into a form of QP optimization problem and then solve it by cvxopt.QP. It has to run almost one night namely 8-9 hours to get the result while SK-learn only needs few minutes. What's more, since I'm still using others' package to solve the convex optimization problem, I don't have a real insight of the whole process. Thus the problem of low accuracy could be the cvxopt package.

V. CNN

A. Theoretical Analysis

A convolutional structure is a significant network layer in the neural network. A convolutional layer h_l is organized into a number of feature maps. Each feature map is also called a channel and is obtained by a filter or a kernel operating on h_{l-1}. That is,

$$s = W \otimes h + b$$

The idea of convolution can be shown directly in the following graph. What's more in CNN, it contains ReLU layers for activation, Max-Pooling layers for down-sampling and softmax layer for classification.

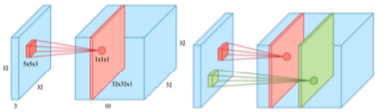

Fig. 4. a visualization of the idea of convolutional neural networks

Here I want to specifically talk about BN(Batch Normalization) since there are an error in my homemade CNN afterwards. Batch Normalization deals with the offseted feature map. It can help us avoid the problem of zero-derivative and do regularization.

B. Experiment

I implemented CNN both by myself(using package numpy) and by Keras(a deep learning platform). Since the task is relatively easy, I use very simple structure by 2 layers of Convolution with BN & ReLU, then two layers of FC and finally go through Softmax to get probability. By the way, the convolution operation is implemented by simple for-loops(I've tried img2col, however I'm too stupid to implement that). The accuracies are shown below.

Method	Accuracy
myCNN	0.8958
Keras(without BN)	0.8958
Keras(with BN)	0.9554

TABLE IV
CNN ACCURACY

First I want to talk about the number 0.8958. It's the accuracy that if a classifier predicts negative for all the samples or say, it just think all the samples are negative. Without Batch Normalization, both my homemade CNN & Keras CNN will get to this number. The reason is that there are too many negative samples in our dataset(9 negative samples per image while 1 positive sample can be generated). Thus the data after some epoches are really biased and the derivative get down to 0 and no more refresh the thing. Finally we will see after softmax classification, the probability for negative sample is 1 and we will see that amazing number.

My CNN implements the homemade Batch Normalization. However maybe there are some bugs inside thus it doesn't change when my CNN adds BN layer.

VI. FACE DETECTION

The idea of face detection here is really simple since we don't use some state-of-art object-detection algorithms like F-RCNN or YOLO. Instead, we use a very simple idea that strides a window along the picture and use our classifier to see if there is a human face.

I choose several size of sliding window to slide over the image in order to get different size of faces. What's more I implement NMS(Non-Maximum Suppression) to make the bounding boxes not so much and get the clear bounding box for the faces as Fig.5.

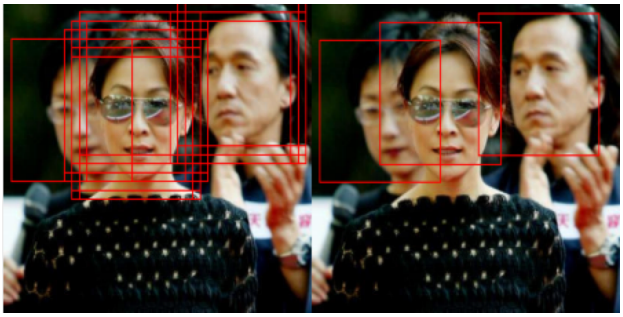

Fig. 5. Leftside : Before NMS, Rightside : After NMS

We can see that NMS did great progress to our detection tasks. Now let's see some other examples of face detection task and we can see some bad examples here. When the environment gets noiser, the classifier can do something wrong, which gives us the incorrect bounding boxes as follow. As shown in Fig.6, there are noise in the whole graph such as the

intricate clothes of the racer and the movement of his hand. Thus our classifier gives another bounding box on the place that it thinks it's another face.

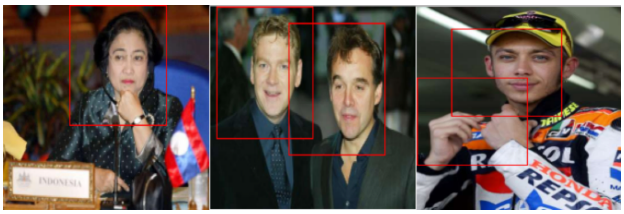

Fig. 6. Some more examples of face detection

Need to say that, due to the in-stability of the model, there are lots of bad samples during detection. Fig.6 shows us some relatively good ones.

Fig.6 shows the detection result using Logistic Regression. Different models are tried via detection. SVM, Logistic Regression and CNN have relatively good result while using LDA gives us many bad samples. We don't put too many images here to show those results. However I did something different in order to get a more accurate result. That is, using multi-model detection during the task. Since there are still bad result using single model, then we use several models to detect. Only when all the models think that there is a face, we give it a bounding box. The detection result is shown in Fig.7.

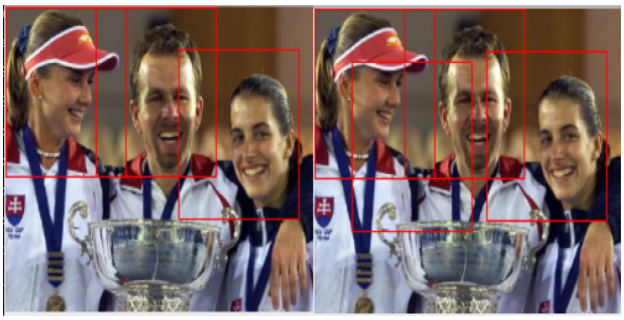

Fig. 7. Multi-Model detection comparison to original one model type

We can see the method did improve the detection results. But it can only do so much to it and for many of the pictures, the models will reach a census on faces classification. Thus the result won't change a lot while using the multi-model detection.

VII. Feature Visualization

A. Theoretical Analysis

For visualization part, I have used t-SNE & PCA both to see their effects. Thus the theoretical part will be seperated to two parts namely the principle of t-SNE & PCA.

1) PCA: PCA, or say Principle Component Analysis, deals with dimension reduction in most of times. It aims to reduce the dimension of the given feature to get simpler but more effective features to go on for our classification or regression process. However think about reduction to 2 dimensions. If there are only 2 dimensions, then we can get the pictures visualized in a simple 2-d plane.

PCA mathematically wants to maximize the possible variance from x, that is

$$w = argmax_{||w||=1} Var_i(t_i) = argmax \frac{w^T X^T X w}{w^T w}$$

Using the lagrange multiplier point of view we can get that $X^T X w = -\lambda w$ which indicates that w is the singular vector of the matrix X or say the eigen vector of matrix $X^T X$. From the equation, we can get PCA done and selecting the dimension=2 gets the visualization done.

2) t-SNE: t-SNE(Stochastic Neighbor Embedding) is a method aims to visualize the data distribution. The central idea of SNE is to then use lower dimensional vectors so as to preserve relationships that are initially measured through the relationship equation between exemplars x_j and x_i that

$$P_{ij} \propto N(x_i, \sigma_i^2)$$

We can initially think about just replacing x_i and x_j with h_i and h_j, and the h_i terms could then be found by solving

$$Loss = KL(P||Q) = \sum_{i,j} P_{i,j} log(\frac{P_{ij}}{q_{ij}})$$

which is simply the Kullback-Leibler divergence from q to p.

What's more, the "t" in t-SNE we use t-distributions instead of normal ones and q_{ij} becomes that

$$q_{ij} = \frac{(1 + |h_j - h_i|^2)^{-1}}{\sum_{m=1}^{n} (1 + |h_m - h_i|^2) - 1}$$

B. Experiment

Note that all the visualization using SK-learn contains all the data. However, the visualization using tensorboard is downsampled to 10000 points to reach faster result. But the result is also almost proper for visualization.

1) PCA: First we deal with the Principle Component Analysis dimension reduction to do visualization. I visualize the PCA both using tensorboard and SK-learn. Method using tensorboard can be 3-d or 2-d. Here we only show those 2-d cluster since 3-d can't be shown in the paper. The PCA visualization result is shown as below. In the picuture, we will see that the result using tensorboard is almost the same as the one using SK-learn, which is our expectation for the visualization.

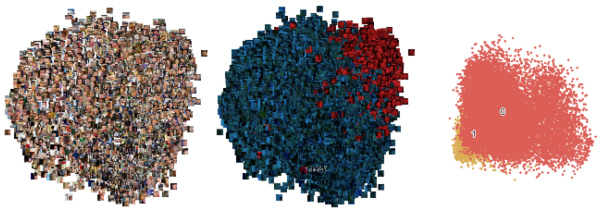

Fig. 8. PCA Visualization of face features(900-d HoG)

In Fig.8, the leftmost two are using tensorboard PCA to visualize. The second graph is colored by label. The rightmost graph is clustered by SK-learn PCA.

2) t-SNE: Also, I visualize t-SNE result both tensorboard and SK-learn. The result is shown below. Need to say that, I colored those face images with labels to make us clearly see the clustering results.

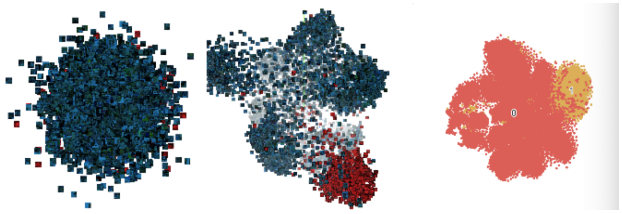

Fig. 9. t-SNE Visualization of face features(900-d HoG)

In Fig.9, the leftmost one is the points before t-SNE training phase. The middle one is tensorboard visualization of t-SNE result and the rightmost one is the SK-learn visualization result.

APPENDIX A
LAGRANGE MULTIPLIERS

Consider a simple optimization problem

$$min_w f(w) s.t. g(w) = 0$$

Let us slightly change w by δw along the curve of $g(w) = 0$, then

$$\forall \delta w, \delta w^T \frac{\partial f}{\partial w} = 0, s.t. \delta w^T \frac{\partial g}{\partial w} = 0$$

otherwise, w does not satisfy the equation above. Thus if we can find a scalar λ that ensures

$$\frac{\partial f}{\partial w} + \lambda \frac{\partial g}{\partial w} = 0, imply\ that \frac{\partial L}{\partial w} = 0$$

$$g(w) = 0, imply\ that \frac{\partial L}{\partial \lambda} = 0$$

then we find a solution to the optimization problem. Thus we optimize $L(w, \lambda) = f(w) + \lambda g(w)$ and ensure that $\frac{\partial L}{\partial w} = 0$, $\frac{\partial L}{\partial \lambda} = 0$

ACKNOWLEDGMENT

Thanks for Prof. Zhang and all the TAs. The course lets me learn a lot deep insight and different perspectives to see machine learning. Without your help, I can't finish all these projects and applications. Thank you.