FUNDAÇÃO GETULIO VARGAS

ESCOLA DE MATEMÁTICA APLICADA

MESTRADO 2015.3

APRENDIZAGEM POR MÁQUINAS

Prof Eduardo Mendes

Projeto Final

KIZZY TERRA

LUCAS MACHADO

RIO DE JANEIRO

DEZEMBRO/2015

1 Introdução

Este relatório visa descrever o processo de implementação das soluções submetidas na Competição de Reconhecimento de Dígitos do Kaggle (https://www.kaggle.com/c/digit-recognizer). O desempenho destas soluçõeos nesta competição será utilizado na disciplina de Aprendizado de Máquina, correspondendo ao seu projeto final, a fim de avaliar a habilidade dos alunos de utilizar técnicas de machine learning para solucionar um problema real.

2 A Competição de Reconhecimento de Dígitos

O objetivo da competição é reconhecer dígitos em imagens que contém dígitos escritos a mão. Os dados para a compentição foram retirados do MNIST dataset. O MNIST ("Modified National Institute of Standards and Technology") dataset é um conjunto de dados clássico dentro da comunidade de Aprendizado de Máquina e tem sido largamente estudado (mais detalhes em http://yann.lecun.com/exdb/mnist/index.html).

3 Pré-processamento dos dados

O pré-processamento dos dados é a primeira etapa (desconsiderando o processo de obtenção) e é importante para conseguir aprender eficientemente o comportamento dos dados. A remoção de dimensões pouco relevantes na aproximação da *target function*, a centralização, que é uma transformação que translada os dados de entrada para a origem removendo viés, e a normalização dos dados, que garante que todas as dimensões estão na mesma escala, são exemplos de pré-processamentos que são importantes de ser aplicados sobre diversos conjuntos de dados antes de utilizá-los em um modelo.

No contexto do problema de reconhecimento de dígitos, cada pixel da imagem é considerado uma dimensão. Com isso, uma imagem 28x28 pixels tem 784 dimensões, e a correlação entre essas dimensões é alta.

Primeiramente, foram retiradas as bordas do conjunto de dados, o qual possui 4 pixels em cada extremo da imagem - inferior, superior, direito e esquerdo -, o que reduziu a quantidade de dimensões para 400. Em algumas imagens, o valor em algumas dimensões na borda não é nulo, o que indica que foi removido ruído das imagens.

Após isso, foram utilizadas quatro técnicas de pré-processamento: centralização, normalização, *data whitening* e *principal component analysis (PCA)*.

A centralização é uma transformação simples e geralmente benigna que translada os dados para a origem removendo qualquer viés dos dados.

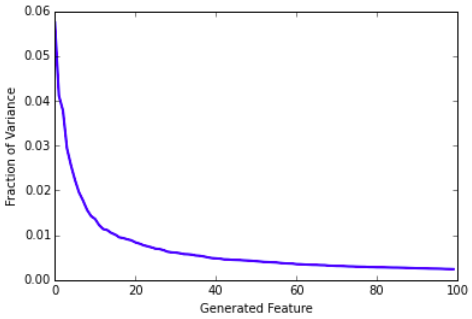
A normalização garante que todas as dimensões estão na mesma escala, o que é uma transformação importante para muitos modelos, como por exemplo os que utilizam a distância euclidiana. Uma motivação a mais que tivemos para normalizar os dados foi a utilização

de *support vector machines (SVM)* com o *radius basis function kernel.* O *rbf kernel* é dado por $K(x, y) = exp(-c * ||x - y||^2)$. Sendo assim, se o dominío das dimensões for muito largo (0-255), então $||x - y||^2$ será muito grande, comprometendo a eficiência do método.

O *data* whitening é utilizado para duas principais finalidades: diminuir a correlação entre as features existentes e forçar todas as *features* a terem a mesma variância. Esta método é bastante utilizado no contexto de classificação de imagens devido a alta redundância de informação presente em pixels adjacentes. No caso de imagens, pixels próximos fornecem informações altamente correlacionadas e o objetivo do *whitening* é diminuir a quantidade de informação redundante.

Por último, a análise de componentes principais, o qual tem como finalidade selecionar as dimensões que possuem a maior influência no processo de generalização. Se uma dimensão possui o valor 0 na maioria das imagens, ele não possui pouca informação sobre o conjunto de dados.

A análise de componentes principais foi feita utilizando-se a classe *decomposition* do pacote *Scikit Learn.* Uma vez feita a decomposição, foi possível gerar um gráfico com a relação entre fração de variância e o número de componentes geradas. A informação obtida com este gráfico acarretou a conlusão de que utilizar de 35 a 40 componentes poderia levar a um resultado satisfatório.




```
Preparing the data...
--------------------------------
Loading the train dataset...
Loading the test dataset...
separate labels from training data
creating PCA object...
fitting the training data...
calculating the amount of variance explained by components...
amount of variance: 0.513882883083
--------------------------------------------------
About 79% of  the total variance in the dataset is explained by 40 compon
ents
Extracting the features...
Shape of training data after dimensionality reduction: (42000, 40)
Transform the test data using existing parameters...
test data shape: (28000, 40)
```

Figura 1: Variância x Número de *features* geradas

4 Seleção de Modelos

A seleção de modelos adequados a serem utilizados na classificação de um determinado conjunto de dados deve considerar os tipos de dados, a quantidade de dados disponível, a complexidade da classificação, a acurácia na classificação e a velocidade de classificação.

No contexto de uma competição do Kaggle, a variável de maior prioridade é a acurácia na classificação uma vez que submete-se o resultado da classificação sobre o conjunto de teste e não o algoritmo implementado em si.

Evidentemente, a acurácia de um classificador baseia-se no conjunto de dados que está sendo analisado, consequentemente esta escolha exige experimentação e por essa razão a

etapa de seleção de modelos consiste de elencar alguns modelos candidatos cuja performance deverá ser comparada para a classificação do conjunto de dados do MNIST. A escolha destes candidatos por sua vez, leva em conta as características e vantagens dos principais modelos conhecidos. A descrição dos classificadores escolhidos se encontram na seção 5.

5 Classificadores Utilizados

Nesta seção descreve-se os modelos selecionados, os quais foram utilizados para a classificação do conjunto de dados do MNIST. A descrição destes algoritmos compreende uma breve análise teórica - incluindo comentários sobre a seleção de parâmetros - e uma análise da respectiva performance obtida.

5.1 Regressão Logística

O primeiro modelo utilizado para classificação foi um modelo de classificação linear: regressão logística. A regressão logística pode ser vista como uma generalização da regressão linear para os problemas de classificação de variáveis categóricas. Uma grande diferença entre os dois modelos está no fato de que a regressão logística prevê a probabilidade de resultados particulares atarvés da função de distribuição de logística. Os valores previstos neste de classificação são, portanto pertencentes ao intervalo $(0, 1)$. Desta forma, a regressão logística prevê a probabilidade de resultados particulares.

A implementação do modelo de Regressão Logística utilizada foi a do pacote *sklearn.linear_model* feito para a linguagem *Python.* A performance obtida com este classificador na submissão no Kaggle foi de $91,00\%$. Este resultado evidencia que para este conjunto de dados um modelo simples já apresenta uma boa capacidade de generalização.

```
print "Preparing the data..."
print "------------------------------"
print "Loading the train dataset..."
mnist_train = pd.read_csv('train.csv')
print "Loading the test dataset..."
test = pd.read_csv("test.csv").values
print "separate labels from training data"
target = mnist_train[[0]].values.ravel()
train = mnist_train.iloc[:,1:].values
```

```
estimator = LogisticRegression()
```

```
print "Training the data..."
print "----------------------------"
estimator.fit(train, target)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, penalty='l2', random_state=None, tol=0.0001)
```

```
predicted = estimator.predict(test)
```

```
predicted
```

```
array([2, 0, 9, ..., 3, 9, 2])
```

```
np.savetxt('submission_logistic_regression.csv', np.c_[range(1, len(test)+1), predicted], delimiter=',', header='ImageI
```

```
Score on Kaggle: 0.91000
```

Figura 2: código Regressão Logística

5.2 k-NN

O k-NN té uma generalização do algoritmo *Nearest Neighbors* o qual consiste de considerar a informação dos pontos vizinhos mais próximos como uma aproximação para um determindado ponto que se quer classificar. Este é um algoritmo simples cuja vantagem é a não exigência de tempo de treinamento, entretanto os requisitos de memória e tempo de classificação são relativamente grandes.

O parâmetro k controla o trade off entre a aproximação e a generalização. Escolhendo$k$ muito pequeno obtém-se uma regra de decisão complexa, o que causa overfitting nos dados; Por outro lado, um k muito garnde conduz a underfitting.

Para a o problema de classificação dos dígitos optou-se por utilizar o modelo com $k = 3$ e a acurácia obtida na submissão foi de 97.70%. Utilizando *cross-validation* foi possível comparar a performance do modelo com $k = 3$ e $k = 5$ como mostra a figura 3.

```
print "Training the data..."
print "----------------------------"

def train_knn(n_neighbors, train_norm, target):
    knn = KNeighborsClassifier(n_neighbors)
    knn.fit(train_norm, target)
    return knn


print "Training with n_neighbors=5 ..."
knn = train_knn(5, train_norm, target)
print "evaluating performance with cross-validation..."
print cross_validation.cross_val_score(knn, train_norm, target, cv=5)

print "-----------------------------------"

print "Training with n_neighbors=3 ..."
knn = train_knn(3, train_norm, target)
print "evaluating performance with cross-validation..."
print cross_validation.cross_val_score(knn, train_norm, target, cv=5)
```

Figura 3: código k-NN

```
Preparing the data...
------------------------------
Loading the train dataset...
Loading the test dataset...
separate labels from training data
creating PCA object...
fitting the training data...
calculating the amount of variance explained by components...
amount of variance: 0.787153004634
-----------------------------------------------
About 79% of  the total variance in the dataset is explained by 40 compon
ents
Extracting the features...
Shape of training data after dimensionality reduction: (42000, 40)
Transform the test data using existing parameters...
test data shape: (28000, 40)
Normalizing the dataset to use nearest neighbors...
Training the data...
----------------------------
Training with n_neighbors=5 ...
evaluating performance with cross-validation...
[ 0.96740036  0.9696537   0.96713895  0.96665476  0.96915198]
-----------------------------------
Training with n_neighbors=3 ...
evaluating performance with cross-validation...
[ 0.96847115  0.97024872  0.96749613  0.96808384  0.9697475 ]
```

Figura 4: Saída do k-NN

5.3 SVM e *kernel*

Dentre os modelos utilizados, o que apresentou o melhor resultado foi *support vector machines* (SVM) com *kernel*. Uma grande vantagem deste modelo é sua robustez a ruído, ou seja, é mais difícil de ocorrer *overfitting*. O *kernel* se constitui de método eficiente para utilizar transformações não-lineares em grandes dimensões. Ou seja, temos um modelo não-linear com regularização automática.

Um modelo SVM é uma representação dos exemplos presentes no conjunto de treinamento como pontos no espaço, mapeados de maneira que os exemplos de cada classe sejam divididos por hiperplanos ótimos.

Dado o pré-processamento descrito na seção 3, foram utilizados 40 componentes a partir do PCA, e o kernel utilizado foi o *radius basis function*. Com isso, dois hiperparâmetros precisamos ser escolhidos: o parâmetro de penalização C e parâmetro *gamma*, que especifica

a “largura” do kernel.

Para a escolha de tais parâmetros foi utilizado o método *cross-validation* com *kfold=3*. O melhor par de parâmetros foi escolhido dentre 15 valores de C gerados aleatoriamentro entre 10^{-2} e 10^{4} e 15 valores para gamma entre 10^{-4}e 10^{3}.

Após 4 horas o algoritmo retornou como melhores hiperparâmetros $C = 2.276$ e $gamma = 0$. O erro *cross-validation* foi 0.01700 e, após a submissão no *kaggle*, o erro *out-of-sample* foi 0.01571.

Este foi o modelo de melhor performance e, consequentemente, o modelo escolhido para a solução do desafio.

```
print "Preparing the data..."
print "------------------------------"

print "Loading the train dataset..."
train_data = np.loadtxt("train_without_target_and_border.csv", delimiter=",")
print "Loading the target dataset..."
target = np.loadtxt("target.csv", delimiter=",")
print "Loading the test dataset..."
test_data = np.loadtxt("test_without_border.csv", delimiter=",")


train_data_normalized = preprocessing.normalize(train_data, norm='l2')
test_data_normalized = preprocessing.normalize(test_data, norm='l2')
#number of components to extract
print "Reduction ..."
pca = PCA(n_components=35, whiten=True)
print "target shape: {0}, train shape: {1}".format(target.shape, train_data_normalized.shape)
print target

pca.fit(train_data_normalized)


print "transform training data..."
train_data_normalized = pca.transform(train_data_normalized)
print "transform test data..."
test_data_normalized = pca.transform(test_data_normalized)

print "Choose best hyperparameters..."
gammas = np.logspace(0.001,0.005,20)
cs = np.logspace(1,4,20)

svc = svm.SVC(kernel='rbf')
clf = grid_search.GridSearchCV(estimator=svc, param_grid=[dict(gamma=gammas), dict(C=cs)],n_jobs=10)
clf.fit(train_data, target)

print "best hyperparameters:\nC:{0}\ngamma:{1}".format(clf.best_estimator_.C, clf.best_estimator_.gamma)

print "apply svm with best hyperparameters"
svc2 = svm.SVC(gamma=clf.best_estimator_.gamma, C=clf.best_estimator_.C)
svc2.fit(train_data_normalized, target)
test_y = svc2.predict(test_data_normalized)
pd.DataFrame({"ImageId": range(1,len(test_y)+1), "Label": map(int,test_y)}).to_csv('hyper_parameters_svm_normalized.csv
```

Figura 5: código SVM

6 Conclusão

O desenvolvimento do Projeto Final proposto permitiu que fossem colocados os aprendizados adquiridos ao longo da disciplina em prática, tendo sido, por esta razão, uma experiência bastante proveitosa e interessante.

De certo modo, o modelo de projeto escolhido - uma competição do Kaggle - é um motivador a mais quando se está buscando boas soluções que permitam obter a melhor performance possível na solução do desafio.

Referências