Trabalho 2

(Descrição do trabalho)

Relatório

Primeiramente foi realizada a normalização do dataset inteiro usando o programa `./svm-scale`. Os dados foram normalizados entre 0 e 1 pois não havia número negativos no dataset.

> ./svm-scale -l 0 -u 1 -s range1 data > data.scale

Após a normalização, o arquivo `data.scale` foi separado em dois arquivos. Foram extraídos 3000 instâncias (~10%) do início e do final do arquivo e criados os arquivos `treino.vet` e `teste.vet`. Foram utilizadas apenas ~10% das instâncias devido ao tempo de processamento.

> head -n 3000 data.scale > treino.vet

> tail -n 3000 data.scale > teste.vet

Foi realizada a busca pelos parâmetros do kernel RBF (g) e da variável de custo (C) utilizando cross validation com o `easy.py`[1].

> python easy.py treino.vet teste.vet

Os valores encontrados para g e C foram:

> Best c=2.0, g=0.03125 CV rate=99.1

A acurácia no treinamento foi de 99.1%.

No teste a acurárica foi de 96.6667%

> Accuracy = 96.6667% (2900/3000) (classification)

Esta diferença na acurácia é explicada, pois o dataset de teste são para instâncias não conhecidas.

O `easy.py` gerou os seguinte arquivos:

- `treino.vet.model`: contém o modelo gerado no treinamento.
- `teste.vet.predict`: contém o resultado das predições com o dataset de teste.
- `treino.vet.scale`: dataset de treinamento normalizado.
- `teste.vet.scale`: dataset de teste normalizado.
- `treino.vet.range`: utilizado pelo easy.py para normalizar os datasets de treino e teste.

A matriz de confusão gerada é (predição x teste):

Classes	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	0	1	1	1	0	0
1	0	614	0	1	0	0	1	0	9	1

Classes	0	1	2	3	4	5	6	7	8	9
2	0	2	297	6	1	0	1	4	2	0
3	0	0	1	279	0	3	0	0	1	2
4	0	0	0	0	284	0	0	2	1	4
5	0	1	0	0	0	280	2	0	1	1
6	0	2	0	0	3	0	292	0	2	0
7	0	0	2	11	0	0	0	300	0	3
8	0	3	1	1	0	0	3	0	273	6
9	0	0	0	0	5	0	0	5	3	281

É possível observar que o fold escolhido para o treinamento do cross validation tem um problema de representatividade. Não há nenhuma instância da classe 0 no intervalo escolhido. Isto pode ser resolvido realizando mais folds.

Depois do cross validation, treinou-se 50% do dataset (`data.scale`) normalizado com os parâmetros c=2.0 e g=0.03125. Primeiro foram gerados os arquivos `data.scale.treino` e `data.scale.teste` com o comando:

```
split -l 29323 data.scale
```

Os dois arquivos gerados foram renomeados para `data.scale.treino` e `data.scale.teste`, respectivamente.

O arquivo `data.scale.treino` foi treinado com o comando:

```
./svm-train -b 1 -c 2 -g 0.03125 data.scale.treino
```

que gerou o arquivo `data.scale.treino.model`.

Após o treinamento foi realizada a predição utilizando o comando:

```
./svm-predict -b 1 data.scale.teste data.scale.treino.model data.scale.teste.predict
```

A acurácia obtida foi de 98.1243%

```
Accuracy = 98.1243% (28773/29323) (classification)
```

Com a seguinte matriz de confusão:

Classes	0	1	2	3	4	5	6	7	8	9
0	2482	0	3	3	2	3	9	0	18	2
1	0	3569	8	0	1	3	6	5	8	3
2	5	12	2899	13	1	0	0	31	10	0
3	0	2	5	2840	0	18	0	10	6	2

Classes	0	1	2	3	4	5	6	7	8	9
4	1	3	2	1	2809	0	3	8	9	20
5	1	1	0	20	0	2721	13	1	16	7
6	9	2	1	0	6	10	2890	0	10	0
7	0	6	12	14	2	2	0	2981	6	18
8	7	4	11	16	2	11	8	0	2738	11
9	0	1	3	3	38	2	0	13	27	2844

A acurárica do cross validation e do teste de validação foi 96.6667% e 98.1243%, respectivamente. Isto evidencia que é possível treinar o modelo com um conjunto menor de dados para obter rapidamente um custo e gama satisfatórios que conseguem manter ou melhorar a taxa de acerto em datasets maiores.

Comparação do Naive Bayes, DT e SVM com relação à probabilidade

O teorema de Bayes fornece uma maneira de calcular a probabilidade a posteriori, $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$. O classificador Naive Bayes assume que o efeito do valor de um preditor (x) em uma determinada classe (c) é independente dos valores de outros preditores.

Para DTs, a probabilidade prevista é a probabilidade média de uma folha, ou a fração de árvores que votam em qualquer classe. A probabilidade de predição para um caso particular é a probabilidade desse caso no nó da árvore usada para predição.

As SVMs não produzem probabilidades nativamente, mas métodos de calibração de probabilidade podem ser usados para converter a saída em probabilidades de classe. Existem vários métodos, incluindo a escala de Platt e regressão isotônica. Se probabilidades forem usadas para medir o desempenho do classificador, isso deve ser feito usando um conjunto de testes independentes para evitar bias.

Comparação com a implementação do scikit-learn

O teste com o dataset reduzido (para cross validation) scikit-learn produziu os seguintes resultados:

> Best c=2.0, g=0.5 CV rate=97.1

E matriz de confusão:

Classes	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	618	2	0	0	0	0	1	1	0
2	0	0	297	1	0	0	0	2	1	0
3	0	0	5	279	0	0	0	13	1	0

Classes	0	1	2	3	4	5	6	7	8	9
4	0	0	1	0	283	0	3	0	1	5
5	0	0	0	2	0	282	0	0	0	0
6	2	1	0	0	1	1	292	0	3	0
7	0	0	6	0	3	0	0	299	0	4
8	0	9	0	1	1	1	1	0	276	3
9	0	2	0	1	1	1	0	2	4	287

A matriz de confusão ficou bastante parecida com o teste com o libsvm, no entanto a taxa de acerto foi ~0.5% maior com o scikit-learn. A melhora com o scikit-learn deve-se a maior quantidade de folds na etapa de cross validation, 3 contra 1 no teste realizado com a libsvm.

Não foi possível finalizar o teste completo com 29323 instâncias utilizando o cross validation do sci-kit devido ao tempo. Porém pode-se inferir que o resultado deverá ser melhor pelo maior número de folds no treinamento. Pretende-se terminar o teste. Os resultados serão publicados neste repositório https://github.com/thild/doc/tree/master/ml/work2.

1 Os arquivos `treino.vet` e `teste.vet` foram copiados para a pasta dos `easy.py`. ↩

Laboratório SVM

1) Baixe e instale a libSVM no seu diretório. Disponível no link abaixo

https://www.csie.ntu.edu.tw/~cjlin/libsvm/

2) Compile o código para gerar os executáveis > make

3) Leia o documento disponível nos links abaixo

http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

4) Utilize a base disponibilizada e separe 50% para treinamento e 50% para teste.

5) Execute o script python que acompanha a libSVM, chamado easy.py (localizado no diretório tools).

Esse script faz a busca pelos parâmetros do kernel RBF (g) e da variável de custo (C). O script gera alguns arquivos. Liste quais são esses arquivos e explique o conteúdo dos mesmos.

> easy.py treino.vet teste.vet

5.1) Reporte a taxa de reconhecimento no arquivo de teste

5.2) Existe diferença com a taxa de reconhecimento no arquivo de treinamento? Se sim, explique porque.

5.3) Com base no arquivo de predição (.predict), monte a matriz de confusão.

6) Utilize os parâmetros encontrados pelo easy.py para treinar um SVM que estima probabilidades.

Para fazer isso você deve usar a opção –b e o programa svm-train. Use também os arquivos normalizados gerados pelo easy.py

```
./svm-train –b 1 –C ?? –g ?? TRAINING-SET modelo
```

7) Utilize o modelo aprendido para estimar probabilidade no passo anterior para classificar a base de teste, estimando probabilidade de cada exemplo de teste.

```
./svm-predict –b 1 TESTING-SET modelo output
```

7.1) Compare a taxa de reconhecimento e a matriz de confusão com os resultados do item 5.

7.2) Como a distribuição de probabilidades do SVM se compara com aquelas dos outros classificadores (Naive Bayes e DT)?

8) Compare os resultados da libSVM com os resultados da implementação do scikit- learn.