Exercícios Práticos: Relatório

Pedro Paulo Vezzá Campos 7538743

1 de dezembro de 2013

1 Exercício 2: PCA

Para este exercício foi proposto que os alunos aplicassem o algoritmo PCA em um *dataset* real, no caso, o conjunto DIGITS. Uma representação gráfica das duas *features* que melhor discriminam as classes do conjunto de dados deve ser apresentada.

Primeiramente, procedeu-se com uma rápida revisão do significado do algoritmo *Principal Component Analysis*. Em suma, o PCA é utilizado para decompor um conjunto de dados multivariado em sucessivas componentes ortogonais duas a duas que apresentam a maior variância possível.

Através do estudo de exemplos do Projeto scikit learn, tais como o disponível em `http://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html#example-decomposition-plot-pca-vs-lda-py` foi possível implementar as tarefas propostas para o exercício. O scikit-learn mostrou-se bastante simples de entender e programar. O exercício completo, exceto a parte 2.5, e com comentários possui menos de 50 linhas.

1.1 Atividade Extra

Foi proposto que o processo fosse repetido para outro conjunto de classes. Seria interessante dividir o *dataset* DIGITS em dois: um com os dígitos de 0 a 4 e outro com os dígitos de 5 a 9.

Neste ponto, houve uma pequena dificuldade com a interface do scikit-learn. A função `datasets.load_digits([n_classes])` permite a restrição da quantidade de classes que vão estar disponíveis para processamento mas não quais classes seriam escolhidas. `datasets.load_digits(5)` retorna sempre o conjunto de dados dos dígitos de 0 a 4.

Para implementar esta divisão e ao mesmo tempo aprender programação Python, buscou-se o código fonte da função `load_digits()`. Uma rápida busca na Internet retornou a sua implementação: `https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/datasets/base.py`. O trecho principal de código que faz a filtragem de quais classes seriam retornadas pela função é

```
if n_class < 10:
    idx = target < n_class
```

```
flat_data, target = flat_data[idx], target[idx]
images = images[idx]
```

A primeira linha é responsável pela "mágica": idx é uma lista de booleanos, indicando para cada linha se ela deve ser mantida ou filtrada no conjunto de dados final. Alterando a linha `idx = target < n_class` para `idx = target >= 5` é possível obter os dados para as classes de números de 5 a 9.

2 Exercício 3: KNN

O algoritmo *k-Nearest Neighbors* é bastante intuitivo e demandou pouca revisão das notas de aula para a interpretação dos seus resultados. O KNN se restringe a classificar instâncias de teste apenas analisando os k vizinhos no conjunto de treinamento mais próximos da instância de teste apresentada. Assim, não há a criação de um modelo interno genérico.

Infelizmente, definir um valor de k ótimo é algo que depende intrinsecamente dos dados. Um valor muito pequeno de k deixa a classificação sucetível a ruídos nos dados de treinamento. Já um k muito grande, torna a superfície de decisão menos precisa.

Para este problema, foi interessante ver a API disponibilizada pelo scikit-learn para o KNN, disponível em `http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`. Além da possibilidade da definição da função de métrica utilizada pelo algoritmo (O padrão é a de Minkowsky) é possível definir uma função de pesos para os pontos de dados. Ao definir o parâmetro `weights = 'uniform'` cada ponto possui um peso uniforme no cálculo, enquanto que `weights = 'distance'` define pesos inversamente proporcionais para cada ponto à distância dele ao ponto de teste.

Testes manuais no tamanho dos conjuntos de treinamento e teste para este algoritmo mostraram como ele é adequado para classificar o *dataset* DIGITS com $k = 7$. Ao separar 5% de dados para treinamento é possível obter uma acurácia de 92%. Com 10% para treinamento a precisão subiu para 95% e com 20% chegou a 99%.

3 Exercício 4: Cross Validation

Cross validation é um tópico muito importante para o campo de Aprendizagem Computacional graças ao problema clássico do *overfitting*. A abordagem deste exercício, o *k-fold cross validation* é a mais simples. O conjunto completo de dados é dividido em k conjuntos, com $k - 1$ destes conjuntos destinados ao treinamento do algoritmo escolhido e 1 conjunto para o teste. A acurácia do algoritmo é então calculada como a média das acurácias ao se variar o conjunto utilizado como teste.

Para este exercício foi escolhido o *Naïve Bayes* pela curiosidade de se estudar o desempenho de um algoritmo que é conceitualmente simples e que no primeiro momento aparentava não ser um bom classificador. Assumir que as *features* são

independentes duas a duas parecia implicar em deturpar a classificação de uma maneira que tornasse o algoritmo inútil para instâncias reais.

Felizmente, o exercício mostrou que este algoritmo, apesar de simples, pode ser bastante eficiente. Para um *5-fold cross-validation*, o *Naïve Bayes* atingiu uma acurácia média de 95%, $\sigma = 3\%$. O código neste exercício também foi muito simplificado pela API do sci-kit learn. Removendo-se os comentários e importações de bibliotecas, esta tarefa foi completa com apenas 4 linhas de código.