SVMs, ensembles e otimização de modelos

Support Vector Machines
Conjuntos de modelos; *random forests*
Otimização de hiperparâmetros
Implementação em python com *scikit-learn*

Support Vector Machines

Máquinas de vetor de suporte

As Máquinas de Vector de Suporte/ *Support Vector Machines* (SVM) foram desenvolvidas por Vapnik no início da década de 90

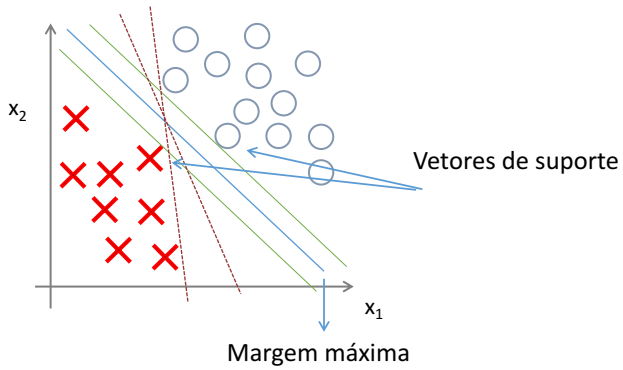
Têm vindo a ganhar popularidade como ferramentas de classificação e regressão, devido a uma série de vantagens teóricas e resultados empíricos

Baseiam-se em 2 ideias:

1 - utilizar somente alguns exemplos de cada classe (vetores de suporte) definindo planos de corte entre classes com margem máxima

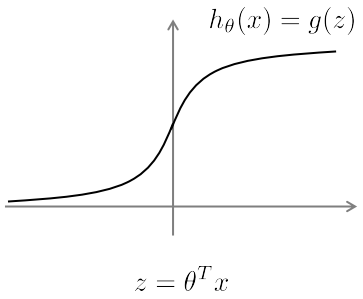
2 - utilizar uma transformação (pode ser não linear) às entradas para um espaço de características lineares, via uma função de Kernel

Classificador com margem máxima



Visão alternativa da regressão logística

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



$y = 1$ queremos $h_\theta(x) \approx 1$ $\theta^T x \gg 0$

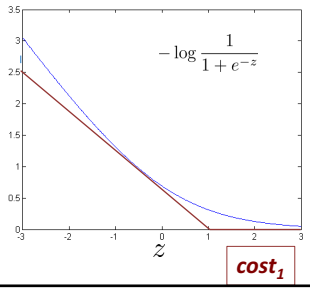
$y = 0$ queremos $h_\theta(x) \approx 0$ $\theta^T x \ll 0$

Visão alternativa da regressão logística

$$-(y \log h_\theta(x) + (1 - y) \log(1 - h_\theta(x)))$$

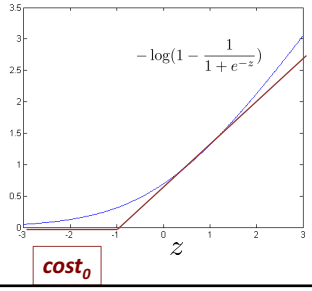
$$= -y \log \frac{1}{1 + e^{-\theta^T x}} - (1 - y) \log(1 - \frac{1}{1 + e^{-\theta^T x}})$$

SE $y = 1$ (queremos $\theta^T x \gg 0$):



$cost_1$

SE $y = 0$ (queremos $\theta^T x \ll 0$):



$cost_0$

Função de custo SVMs

Soft margin SVM: usado nos casos reais onde a separação perfeita não é possível

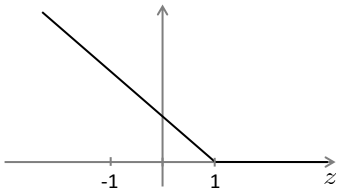
$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

Parâmetro C:
Controlo trade-off
entre erro e complexidade

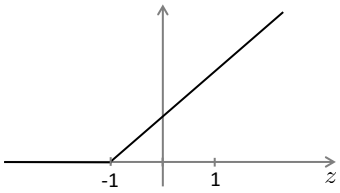
Erro

Regularização

Se y = 1 queremos: $\theta^T x \geq 1$
não apenas >= 0



Se y = 0 queremos: $\theta^T x \leq -1$
não apenas < 0



Kernels

Definem mapeamentos, possivelmente não lineares, para as variáveis (atributos) originais criando "novos" atributos

Dadas variáveis originais, calcular novos atributos usando funções de similaridade com pontos pré-definidos

Em SVMs, estes pontos pré-definidos são os próprios exemplos

Exemplo: kernel gaussiano

$$f_i = \text{similarity}(x, l^{(i)})$$

$$= \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2} \right)$$

SVMs em classificação

O algoritmo de treino SMO garante que é atingido sempre o hiperplano óptimo de separação entre classes.

Existem diversos tipos de kernels (linear, polinomial, gaussiano, splines), sendo o gaussiano (RBF) o mais popular;

Dois parâmetros de configuração: C > 0 (complexidade vs erro) e gamma > 0 (o parâmetro do kernel gaussiano)

- C grande e gamma pequeno favorecem overfitting
- C pequeno e gamma grande podem conduzir a underfitting

Conjuntos de modelos

Conjuntos de modelos

Ideia: resposta a novas situações obtida a partir da combinação das respostas de vários modelos distintos

Componentes:

Conjunto de *L* modelos distintos (de classificação ou regressão)

Função de combinação *c* que, dadas as saídas dos modelos, retorna um valor único de saída

Tarefas:

Como construir o conjunto de modelos ?

Que função de combinação usar ?

Conjuntos de modelos: motivação

Estatística:

normalmente nº de exemplos de treino não é suficiente;

várias alternativas possíveis para os explicar

Computacional:

métodos de construção dos modelos – optimização local; vários modelos possíveis como resultado

Representação:

espaço de representação pode não corresponder ao espaço "real" dos dados; ao considerar vários modelos "expande-se" o espaço de representação

Requisitos para bons resultados:

- Modelos individuais precisos
- Modelos diversos/ heterogéneos

Funções de combinação

Classificação:

Votação: saída é a classe que reúne mais "votos"

Confiança: saída é decidida pelo modelo que apresente maior confiança associada à classe que propõe

Estes dois métodos poderão ser combinados numa votação pesada por confiança.

Regressão:

Média simples: saída é a média das saídas dos modelos

Média pesada: pesos de cada modelo poderão ser calculados com base num processo de estimação de erro prévio, e.g. com exemplos de validação

Construção dos conjuntos de modelos

Abordagem 1: manipular os **exemplos de treino** de formas distintas para criar modelos diferentes:

Bagging: baseado no bootstrap; cada amostra criada por um processo de bootstrap distinto

Validação cruzada: cada modelo cria-se ignorando uma das k partições (tal como no processo anterior)

Boosting:

- cada exemplo terá uma probabilidade de ser escolhido;
- é realizada a escolha com substituição (tipo boostrap) mas considerando as probabilidades;
- ao fim de criar cada conjunto de exemplos, probabilidades são actualizadas diminuindo-se as probabilidades dos exemplos correctamente classificados e aumentando as restantes.

Construção dos conjuntos de modelos

Abordagem 2: **injectar aleatoriedade** no algoritmo

Mudar decisões determinísticas para estocásticas

Escolha atributo a testar na raíz de uma árvore ou na globalidade dos atributos a testar em toda a árvore

Nº de neurónios intermédios de uma RNA

...

Exemplo: ***Random forests***

- Ensemble de árvores de decisão
- Escolha aleatória do conjunto de atributos a testar
- Utilização do bagging como forma de escolher os exemplos
- Algoritmo de indução de árvores: CART
- Usa o erro out-of-bag (nos exemplos não escolhidos pelo bagging) em cada iteração

Construção dos conjuntos de modelos

Gradient boosting

- Ensemble de modelos simples
- Em cada iteração tenta criar um modelo F_{m+1} que é treinado a partir dos resíduos de F_m i.e. treinado para aprender os erros : $y - F_m(x)$
- Valores de previsão são dados pela soma das previsões dos vários modelos
- Pode ser aplicado a vários tipos de modelos, sendo bastante popular com árvores de decisão
- Normalmente usado com “shrinkage”, i.e. um factor multiplicativo que reduz as contribuições de cada modelo para evitar sobre-ajustamento

Seleção de atributos

Seleção de atributos

Em muitos casos práticos há necessidade/ diversas vantagens em **reduzir o nº de atributos de entrada** de um modelo:

- **complexidade** dos modelos aumenta com o nº atributos de entrada o que pode provocar **sobre-ajustamento**;
- dados e modelos podem ser mais facilmente **analisados** e **compreendidos**;
- eliminação de atributos redundantes ou contraditórios pode melhorar o próprio processo de aprendizagem reduzindo o **ruído**

Processo de escolha do melhor sub-conjunto de atributos de um dado conjunto é um **problema de optimização** que pode tornar-se complexo, dado o espaço alargado de procura

Seleção de atributos: estratégia

Algoritmos de **filtragem**:

- Seleção de atributos realizada antes do processo de aprendizagem, de forma independente dos classificadores
- Avaliação dos subconjuntos de atributos realizada com medidas estatísticas (e.g. Entropia/ ganho, etc)

Algoritmos **envolventes** (*wrappers*):

- Seleção dos atributos realizada em paralelo com a construção do modelo
- Avaliação dos subconjuntos de atributos realizada treinando um modelo e estimando o seu erro (usando os métodos estudados)

Algoritmos **embebidos** (*embedded*):

- Seleção dos atributos realizada junto com o processo de aprendizagem (e.g. regressão linear com regularização)

Seleção de atributos: algoritmos de optimização

Heurísticas:

- **Forward selection**: inicia com poucos atributos (e.g. 1) e vai adicionando atributos até atingir um comportamento satisfatório
- **Backward selection**: inicia com todos os atributos e vai removendo

Meta-heurísticas:

- **Algoritmos Evolucionários**
- **Simulated Annealing**

Optimização de modelos/ hiper-parâmetros

Otimização de modelos

De forma a optimizar o processo de construção, em muitos casos, é realizado um processo de otimização de modelos, realizado pela otimização de hiper-parâmetros – parâmetros não otimizados no processo de treino do modelo

- Processo de optimização minimizando-se uma medida de erro sobre um conjunto de exemplos de validação (não usados para o processo de treino) podendo usar-se processos de validação cruzada
- Um dos objetivos é procurar modelos que minimizem função de erro
- Podem ser usadas técnicas semelhantes à seleção de atributos (heurísticas e meta-heurísticas), sendo as mais usadas a procura exaustiva de todas as combinações (grid-search) ou uma procura aleatória (se o espaço de combinações a testar for muito grande)

Otimização de modelos - exemplos

Pruning de árvores de decisão – parâmetros usados

Parametrização de um SVM (C, gamma)

Otimização do parâmetro de regularização em regressão linear

Otimizar parâmetros numa random forest – nº árvores, nº atributos a usar, profundidade da árvore, etc.

Procura da melhor topologia para uma rede neuronal (e.g. nº nós intermédios)

Procura do melhor conjunto e ordem de regras numa lista de decisão

Exemplos com scikit-learn

IMPLEMENTAÇÃO EM PYTHON

Modelos de classificação: árvores

O modelo de árvores de decisão pode ser construído a partir da classe *DecisionTreeClassifier*

```
from sklearn import tree

tree_model = tree.DecisionTreeClassifier()
tree_model = tree_model.fit(train_in, train_out)
print(tree_model)
print(tree_model.predict(test_in))

print("Valores previstos: ”,
tree_model.predict(test_in))
print("Valores reais: ”, test_out)
```

```
from sklearn import datasets
import numpy as np

iris = datasets.load_iris()
indices = np.random.permutation(len(iris.data))
train_in = iris.data[indices[:-10]]
train_out = iris.target[indices[:-10]]
test_in = iris.data[indices[-10:]]
test_out = iris.target[indices[-10:]]
```

Podem ser usadas para regressão
User guide - Section 1.10

IMPLEMENTAÇÃO EM PYTHON

Modelos de regressão linear

Regularização: Lasso e Ridge

```
from sklearn import datasets
diabetes = datasets.load_diabetes()
X_train = diabetes.data[:-20]
X_test = diabetes.data[-20:]
y_train = diabetes.target[:-20]
y_test = diabetes.target[-20:]
```

```
from sklearn import linear_model
ridge = linear_model.Ridge(alpha=.1)
ridge = ridge.fit(X_train, y_train)
print("Valores previstos: ” , ridge.predict(X_test))
```

```
lasso = linear_model.Lasso()
lasso = lasso.fit(X_train, y_train)
print("Valores previstos: " , lasso.predict(X_test))
```

User guide
Section 1.1

Ensembles

IMPLEMENTAÇÃO EM PYTHON

Exemplos de bagging

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.model_selection import cross_val_score

bagged_model = BaggingClassifier(KNeighborsClassifier(), max_samples=0.5, max_features=0.5)

scores_bag = cross_val_score(bagged_model, iris.data, iris.target, cv = 5)
print (scores_bag)
```

```
from sklearn import tree
bagged_model2 = BaggingClassifier(tree.DecisionTreeClassifier(), max_samples=0.5, max_features=0.5)
scores_bag2 = cross_val_score(bagged_model2, iris.data, iris.target, cv = 5)

print(scores_bag2)
```

Ensembles

IMPLEMENTAÇÃO EM PYTHON

Exemplo de random forest

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100)

scores_rf = cross_val_score(rf_model, iris.data,
                    iris.target, cv = 5)

print (scores_rf)
print (scores_rf.mean())
```

Ensembles

IMPLEMENTAÇÃO EM PYTHON

Exemplo de boosting

```
from sklearn.ensemble import AdaBoostClassifier

ada_tree = AdaBoostClassifier(n_estimators=100)

scores_ada = cross_val_score(ada_tree, iris.data, iris.target, cv = 5)

print (scores_ada)
print (scores_ada.mean())
```

AdaBoostRegressor – permite usar em regressão
Parâmetro base_estimator = permite definir outros modelos

Ensembles

IMPLEMENTAÇÃO EM PYTHON

Exemplo de gradient boosting

```
from sklearn import ensemble
from sklearn.metrics import mean_squared_error

params = {'n_estimators': 500, 'max_depth': 4, 'min_samples_split': 2,
        'learning_rate': 0.01, 'loss': 'ls'}
clf = ensemble.GradientBoostingRegressor(**params)
clf.fit(X_train, y_train)
mse = mean_squared_error(y_test, clf.predict(X_test))
print("MSE: %.1f" % mse)

scores_mse = cross_val_score(estimator = clf, X= diabetes.data, scoring = "r2",
                             y = diabetes.target, #score_func = mean_absolute_error, cv = 5)
print(scores_mse)
print(scores_mse.mean())
```

Seleção de atributos

IMPLEMENTAÇÃO EM PYTHON

Filtros por variabilidade – remover atributos que variam "pouco" (abaixo de um threshold definido)

```
from sklearn import datasets, svm
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import cross_val_score

digits = datasets.load_digits()
print (digits.data.shape)
sel = VarianceThreshold(threshold=20)
filt = sel.fit_transform(digits.data)
print (filt.shape)
svm_mod = svm.SVC(gamma=0.001, C=100.)
scores= cross_val_score(svm_mod, digits.data, digits.target, cv= 10)
print (scores.mean())
scores_vt= cross_val_score(svm_mod, filt, digits.target, cv= 10)
print (scores_vt.mean())
```

Experimente variar o threshold !

Ver secção 1.13.1 do *User Guide*

Seleção de atributos

IMPLEMENTAÇÃO EM PYTHON

Filtros por análise univariada (testes estatísticos) – manter atributos com valores melhores de p-value

```
from sklearn.feature_selection import SelectKBest, chi2, f_classif

filt_kb = SelectKBest(chi2, k=32).fit_transform(digits.data, digits.target)
print (filt_kb.shape)
scores_kb = cross_val_score(svm_model, filt_kb, digits.target, cv = 10)
print (scores_kb.mean())

filt_kb2 = SelectKBest(f_classif, k=32).fit_transform(digits.data, digits.target)
scores_kb2 = cross_val_score(svm_model, filt_kb2, digits.target, cv = 10)
print (scores_kb2.mean())
```

Ver secção 1.13.2 do *User Guide*

Seleção de atributos

IMPLEMENTAÇÃO EM PYTHON

Wrapper: *recursive feature elimination (RFE)*

```
from sklearn.feature_selection import RFE

svm_model = svm.SVC(kernel = "linear", C=100.)

rfe = RFE(estimator=svm_model, n_features_to_select=32, step=1)

scores_rfe = cross_val_score(rfe, digits.data, digits.target, cv = 10)

print (scores_rfe.mean())
```

Ver secção 1.13.3 do *User Guide*

Seleção de modelos

IMPLEMENTAÇÃO EM PYTHON

Procura em grelha de parâmetros de SVMs (com validação cruzada na estimação do erro)

```
from sklearn import svm, grid_search, datasets
from sklearn.model_selection import cross_val_score

parameters = {'kernel':('linear', 'rbf'), 'C':[1, 3, 10, 100],
                    'gamma':[0.01, 0.001]}
svm_model_d = svm.SVC( )

opt_model_d = grid_search.GridSearchCV(svm_model_d, parameters)
opt_model_d.fit(digits.data, digits.target)
print (opt_model_d.best_estimator_)
scores_gs = cross_val_score(opt_model_d, digits.data, digits.target, cv = 5)
print (scores_gs.mean())
```

Ver secção 3.2 do *User Guide*

Seleção de modelos

IMPLEMENTAÇÃO EM PYTHON

Procura aleatória (com validação cruzada na estimação do erro)

```
from sklearn.grid_search import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100)
param_dist = {"max_depth": [2, 3, None], "max_features": [2,4,6],
  "min_samples_split": [2,4,6], "min_samples_leaf": [2,4,6],
  "bootstrap": [True, False], "criterion": ["gini", "entropy"]}
rand_search = RandomizedSearchCV(rf_model,
                          param_distributions=param_dist, n_iter=20)
rand_search.fit(digits.data, digits.target)
print (rand_search.best_estimator_)
report(rand_search.grid_scores_)
scores_rs = cross_val_score(rand_search, digits.data, digits.target, cv = 5)
print (scores_rs.mean())
```

Ver secção 3.2.2 do *User Guide*

Seleção de modelos

IMPLEMENTAÇÃO EM PYTHON

Procura aleatória (com validação cruzada na estimação do erro)

```
from operator import itemgetter
import numpy as np

def report(grid_scores, n_top=5):
  top_scores = sorted(grid_scores, key=itemgetter(1), reverse=True)[:n_top]
  for i, score in enumerate(top_scores):
    print("Model with rank: {0}".format(i + 1))
    print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  score.mean_validation_score,
                  np.std(score.cv_validation_scores)))
    print("Parameters: {0}".format(score.parameters))
    print("")
```