MO444 – Aprendizado de Máquina

Edgar Rodolfo Quispe Condori - RA 192617

May 25, 2017

Questão 1. *Processamento de texto*

Para o preprocessamento foi usada a biblioteca *NLTK*

```
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from nltk.tokenize import RegexpTokenizer
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

from nltk.stem import WordNetLemmatizer
'''
This class will do the steming , stopwords removal, tokenization and puntuation points removal
'''
class Stemmer_Tokenizer(object):
    def __init__(self):
        self.wnl = PorterStemmer() #init stemer
    def __call__(self, doc):
        tokenizer = RegexpTokenizer(r'\w+') # init tokenizer
        tokens = tokenizer.tokenize(doc) #remove puntuation points and tokenize
        #stemming and delete stopwords
        return [self.wnl.stem(t) for t in tokens if not t in stopwords.words('english')]

def preprocess():
    #load files
    documents = load_files(container_path = 'filesk', encoding = 'utf8')
    #binary bag of words handler (includes lowercase convertion and low/frequecy words removal)
    bin_vect = CountVectorizer(tokenizer=Stemmer_Tokenizer(), strip_accents = 'ascii',
    lowercase = True, max_df = 4999, min_df = 2, binary = True)
    #term frequency bag of word handler  (includes lowercase convertion and low/frequecy words removal)
    frec_vect = TfidfVectorizer(tokenizer=Stemmer_Tokenizer(), strip_accents = 'ascii',
    lowercase = True, max_df = 4999, min_df = 2)
    #split documents for test and training
    x_train, x_test, y_train, y_test = train_test_split(documents.data, documents.target,
     test_size=1000)
    #compute binary bag of words
    x_bin_train = bin_vect.fit_transform(x_train)
    x_bin_test = bin_vect.transform(x_test)
    #compute term frequency bag of words
    x_frec_train = frec_vect.fit_transform(x_train)
    x_frec_test = frec_vect.transform(x_test)

    return x_bin_train, x_bin_test, x_frec_train, x_frec_test, y_train, y_test

if __name__ == '__main__':
    #preprocess data
    x_bin_train, x_bin_test, x_frec_train, x_frec_test, y_train, y_test = preprocess()
```

Questão 2. *Rode o naive bayes (BernoulliNB) na matriz binaria. Qual a acuracia?*

A acurácia máxima foi de 0.804 com o hiperparámetro $'alpha'$: 0.0009765625
O código:

```
def evaluate_BernoulliNB(x_train, y_train, x_test, y_test):
    print "BernoulliNB"
    #params for gridSearch
    params = {'alpha' : [2**(-15), 2**(-10), 2**(-5), 2**(0), 2**(5)]}
    #gridSearch with 3 folds
    gridSearch = GridSearchCV(BernoulliNB(), params, cv = 3)
    gridSearch.fit(x_train, y_train)
    #train classifier with best params
    clf = BernoulliNB(**gridSearch.best_params_).fit(x_train, y_train)
    print "Params", gridSearch.best_params_
    print "Accuracy", clf.score(x_test, y_test)

if __name__ == '__main__':
    #preprocess data
    x_bin_train, x_bin_test, x_frec_train, x_frec_test, y_train, y_test = preprocess()

    #evaluate BernoulliNB
    evaluate_BernoulliNB(x_bin_train, y_train, x_bin_test, y_test)
```

Questão 3. *Rode o naive Bayes (MultinomialNB) na matriz de term-frequency. Qual a accuracia (compare com a anterior).*

A acurácia máxima foi 0.829 com o hiperparâmetro $'alpha'$: 0.03125. Comparando ele como o BernoulliNB, o MultinomialNB é melhor por 0.025.

O código:

```
def evaluate_MultiNB(x_train, y_train, x_test, y_test):
    print "MultiNB"
    #params for gridSearch
    params = {'alpha' : [2**(-15), 2**(-10), 2**(-5), 2**(0), 2**(5)]}
    #gridSearch with 3 folds
    gridSearch = GridSearchCV(MultinomialNB(), params, cv = 3)
    gridSearch.fit(x_train, y_train)
    #train classifier with best params
    clf = MultinomialNB(**gridSearch.best_params_).fit(x_train, y_train)
    print "Params", gridSearch.best_params_
    print "Accuracy", clf.score(x_test, y_test)


if __name__ == '__main__':
    #preprocess data
    x_bin_train, x_bin_test, x_frec_train, x_frec_test, y_train, y_test = preprocess()

    #evaluate MultiNB
    evaluate_MultiNB(x_frec_train, y_train, x_frec_test, y_test)
```

Questão 4. *Rode o PCA e reduza o numero de dimensoes da matriz de term-frequency para 99% da variancia original. Voce nao consiguira usar o PCA tradicional do Sklearn*

Foi implementada a função *number_Component* que recebe a matriz de dados de treinamento do term frequency e retorna o PCA ajustado com o numero de componentes que mantem o 0.99% da varianza. O número de componentes que manten o 0.990005336813% da varianza é 3018.

O código:

```
def number_Component(x_frec_train):
    #compute pca with max number of components
    pca = TruncatedSVD(n_components = x_frec_train.shape[1] - 1)
    pca.fit(x_frec_train)
    variance = 0.0
    for i in range(len(pca.explained_variance_ratio_)):
        #accumulate the variance
        variance += pca.explained_variance_ratio_[i]
        if variance >= 0.99:
            print "pca result", i, variance
            #refit pca with the correct n_components
            pca = TruncatedSVD(n_components = i).fit(x_frec_train)
            print pca.explained_variance_ratio_.sum()
            return pca
    return pca

if __name__ == '__main__':
    #preprocess data
    x_bin_train, x_bin_test, x_frec_train, x_frec_test, y_train, y_test = preprocess()

    #fit pca with the 0.99 of variance
    pca = number_Component(x_frec_train)

    #apply pca to data
    x_frec_train = pca.transform(x_frec_train)
    x_frec_test = pca.transform(x_frec_test)
```

Questão 5. *Rode SVM com RBF (modo one vs all), gradient boosting e random forest na matriz com o numero de dimensoes reduzidas. Não se esqueca de fazer a busca de hiperparametros. Quais as acurácias?*

A avaliação do SVM mostra uma acurácia de 0.844 com o hiperparámetro $'C' : 32,'gamma' : 0.03125$.

O código:

```
def svm_evaluation(x_train, y_train, x_test, y_test):
    print "Support Vector Machine"
    #params for gridSearch
    params = {'C' : [2**(-5), 2**(0), 2**(5), 2**(10)], 'gamma': [2**(-15), 2**(-10),
      2**(-5), 2**(0), 2**(5)]}
    #gridSearch with 3 folds
    gridSearch = GridSearchCV(SVC(random_state = 1, kernel = 'rbf', decision_function_shape = 'ovr'),
      params, cv = 3, n_jobs = -1)
    gridSearch.fit(x_train, y_train)
    #train classifier with best params
    clf = SVC(**gridSearch.best_params_).fit(x_train, y_train)
    print "Params", gridSearch.best_params_
    print "Accuracy", clf.score(x_test, y_test)

if __name__ == '__main__':
    #preprocess data
    x_bin_train, x_bin_test, x_frec_train, x_frec_test, y_train, y_test = preprocess()

    #fit pca with the 0.99 of variance
    pca = number_Component(x_frec_train)

    #apply pca to data
    x_frec_train = pca.transform(x_frec_train)
    x_frec_test = pca.transform(x_frec_test)

    #evaluate svm
    svm_evaluation(x_frec_train, y_train, x_frec_test, y_test)
```

A avaliação do Gradient Boosting Machine mostra uma acurácia de 0.834 com o hiperparámetro $'n\_estimators' : 100,'learning\_rate' : 0.1,'max\_depth' : 5$.

O código:

```
def gb_evaluation(x_train, y_train, x_test, y_test):
    print "Gradient Boosting Machine"
    #params for gridSearch
    params = {'n_estimators' : [30, 70, 100], 'learning_rate' : [0.1, 0.05], 'max_depth': [5]}
    #gridSearch with 3 folds
    gridSearch = GridSearchCV(GradientBoostingClassifier(random_state = 1), params,
      cv = 3, n_jobs = -1)
    gridSearch.fit(x_train, y_train)
    #train classifier with best params
    clf = GradientBoostingClassifier(**gridSearch.best_params_).fit(x_train, y_train)
    print "Params", gridSearch.best_params_
    print "Accuracy", clf.score(x_test, y_test)

if __name__ == '__main__':
    #preprocess data
    x_bin_train, x_bin_test, x_frec_train, x_frec_test, y_train, y_test = preprocess()
```

```python
    #fit pca with the 0.99 of variance
    pca = number_Component(x_frec_train)

    #apply pca to data
    x_frec_train = pca.transform(x_frec_train)
    x_frec_test = pca.transform(x_frec_test)

    #evaluate GradientBoostingClassifier
    gb_evaluation(x_frec_train, y_train, x_frec_test, y_test)
```

A avaliação do Random Forest mostra uma accurácia de 0.724 com o hiperparámetro $'max\_features'$: 60,$'n\_estimators'$: 800.

O código:

```python
def rf_evaluation(x_train, y_train, x_test, y_test):
    print "Random Forest"
    #params for gridSearch
    params = {'n_estimators' : [100, 200, 400, 800], 'max_features': [50, 54, 60]}
    #gridSearch with 3 folds
    gridSearch = GridSearchCV(RandomForestClassifier(random_state = 1), params,
      cv = 3, n_jobs = -1)
    gridSearch.fit(x_train, y_train)
    #train classifier with best params
    clf = RandomForestClassifier(**gridSearch.best_params_).fit(x_train, y_train)
    print "Params", gridSearch.best_params_
    print "Accuracy", clf.score(x_test, y_test)

if __name__ == '__main__':
    #preprocess data
    x_bin_train, x_bin_test, x_frec_train, x_frec_test, y_train, y_test = preprocess()

    #fit pca with the 0.99 of variance
    pca = number_Component(x_frec_train)

    #apply pca to data
    x_frec_train = pca.transform(x_frec_train)
    x_frec_test = pca.transform(x_frec_test)

    #evalutate RandomForestClassifier
    rf_evaluation(x_frec_train, y_train, x_frec_test, y_test)
```

Questão 6. *Qual o melhor classificador dos testados?*

Os resultados para cada um dos algoritmos são:

Algoritmo	**Accuracy**
BernoulliNB	0.804
MultiNB	0.829
Support Vector Machine	0.844
Random Forest	0.724
Gradient Boosting Machine	0.834

O melhor classificador é o SVM que tem uma differencia de 0.010 como o segundo melhor classificador (GBM), o interesante é o resultado do MultiNB e BernoulliNB que têm uma accuracia acima de 0.8.

Treinando o melhor classificador tem-se uma acurácia de 0.8382 com os parámetros $'C'$: 32,$'gamma'$: 0.03125.

O código:

```python
def best_classifier(x, y):
    print "Best classifer"
    #params for gridSearch
    params = {'C' : [2**(-5), 2**(0), 2**(5), 2**(10)], 'gamma': [2**(-15), 2**(-10),
      2**(-5), 2**(0), 2**(5)]}
    #gridSearch with 3-folds
    gridSearch = GridSearchCV(SVC(random_state = 1, kernel = 'rbf', decision_function_shape = 'ovr'),
      params, cv = 3, n_jobs = -1)
    gridSearch.fit(x, y)
    #print best results
    print "Best Params", gridSearch.best_params_
    print "Best Accuracy", gridSearch.best_score_
    #create final classifier with best params
    return SVC(**gridSearch.best_params_).fit(x, y)


if __name__ == '__main__':
    #preprocess data
    x_bin_train, x_bin_test, x_frec_train, x_frec_test, y_train, y_test = preprocess()

    #fit pca with the 0.99 of variance
    pca = number_Component(x_frec_train)

    #apply pca to data
    x_frec_train = pca.transform(x_frec_train)
    x_frec_test = pca.transform(x_frec_test)

    #train best classifier
    best_classifier(np.vstack((x_frec_train, x_frec_test)),
      np.vstack((y_train.reshape((-1, 1)), y_test.reshape((-1, 1)) )).reshape(-1))
```