Projeto Prático 02: *Dogs vs. Cats*

Leyza Baldo Dorini

Este relatório discute de forma sucinta os resultados de classificação obtidos para a base de dados *Dogs vc Cats*, disponibilizada na plataforma Kaggle[1]. A base de treinamento possui 25000 imagens e a de testes 12500, ambas compostas por imagens de gatos e cachorros. A Figura 2 ilustra algumas amostras extraídas da base de treinamento. Como pode-se observar, a complexidade é grande, tendo impacto nas taxas de classificação. Estão presentes, dentre outros fatores, oclusão parcial do objeto de interesse, presença de mais de um animal na mesma imagem, dimensões diferentes e imagens sintéticas.

Figura 1: Imagens extraídas da base de dados *Dogs vs Cats*.

Para o desenvolvimento deste trabalho, foi composta uma base auxiliar com 500 imagens extraídas de forma aleatória da base de treinamento. Tal base auxiliar foi dividida de tal forma que 75% das imagens foram utilizadas para treinamento e 25% para testes.

Os testes experimentais consideraram os três classificadores abordados até o momento na disciplina: árvores de decisão, Naive Bayes e k-NN. Os parâmetros foram determinados com base na função `RandomizedSearchCV`, a qual busca os parâmetros que conduzem a melhores resultados de classificação com base em validação cruzada. Para árvores de decisão, os parâmetros e a faixa de valores considerados foram:

```
param_dist = {"criterion": ["gini", "entropy"],
              "min_samples_split": randint(5, 20),
              "max_depth": randint(1, 20),
              "min_samples_leaf": randint(1, 20),
              "max_leaf_nodes": randint(2, 20)}
```

Para o algoritmo k-NN, variou-se a quantidade de vizinhos entre 1 e 21. A classificação baseada na abordagem Naive Bayes considerou os parâmetros *default* da implementação disponível no `scikit-learn`.

Na escolha dos descritores, procurou-se explorar diferentes características de forma e cor. Além das abordagens disponibilizadas junto com o projeto (utilização das intensidades da imagem em níveis de cinza - *raw* - e histograma de cores no espaço HSV), foram implementados também SIFT, DAISY, HOG e Sobel. Os parâmetros de cada descritor estão descritos no código fonte disponibilizado junto a este relatório.

[1] https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition

1 Metodologia

Inicialmente, foram selecionadas 500 imagens da base de treinamento, sendo metade de cada classe. Cada imagem foi processada de tal forma a ficar com a mesma dimensão (128×128 pixels). Apesar de necessário para o funcionamento de diversas abordagens, esse processo pode introduzir problemas. Por exemplo, imagens com proporção 128×1000 podem perder características de textura e forma, essenciais para o processo de classificação.

Posteriormente, foram extraídos todos os descritores para cada imagem, os quais foram utilizados de forma individual nos três algoritmos de classificação considerados.

As medidas utilizadas para comparação dos resultados foram a acurácia e f1-*measure* para as imagens de teste da base auxiliar criada para o desenvolvimento do projeto.

2 Resultados

A Tabelas 1 e 2 detalham, respectivamente, as taxas de acurácia e f1-*measure* obtidas para a base testes. Os valores em negrito representam as melhores taxas para cada descritor. A última linha sumariza a melhor taxa para cada classificador.

Os parâmetros escolhidos para cada classificador estão listados no Anexo 2. Como mencionado anteriormente, eles foram determinados usando a função `RandomizedSearchCV`.

Tabela 1: Acurácia para classificação na base de testes

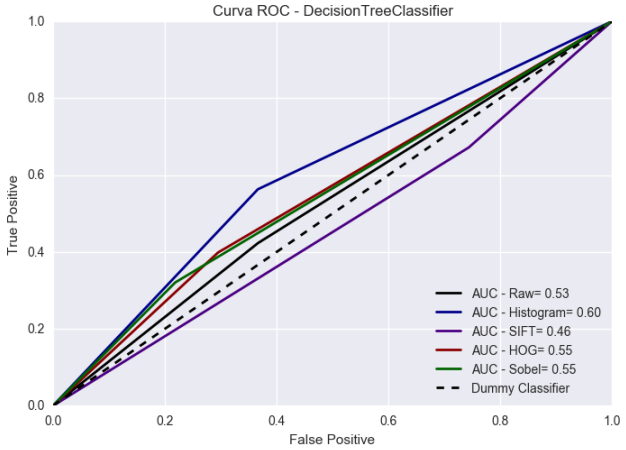
	Árvores de Decisão	Naive Bayes	k-NN
Intensidades			
Histograma de cores			
SIFT			
DAISY			
HOG			
Sobel			
Melhor taxa			

Tabela 2: f1-*measure* para classificação na base de testes

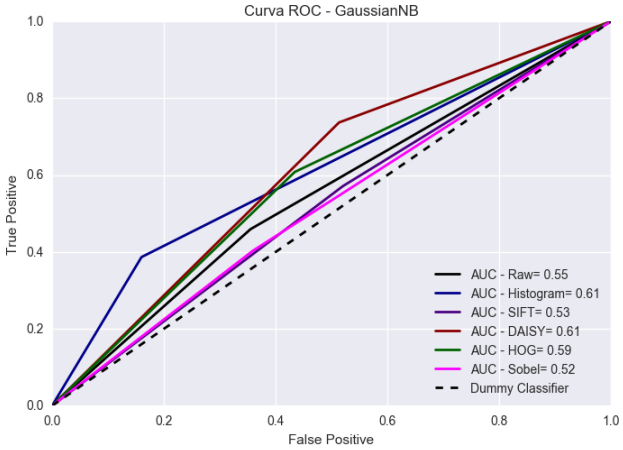
	Árvores de Decisão	Naive Bayes	k-NN
Intensidades			
Histograma de cores			
SIFT			
DAISY			
HOG			
Sobel			
Melhor taxa			

Observe que.... Discutir os resultados de forma sucinta e procurando explorar pontos que não são necessariamente óbvios, fazendo relação entre diferentes fatores, procurando argumentos para afirmações....

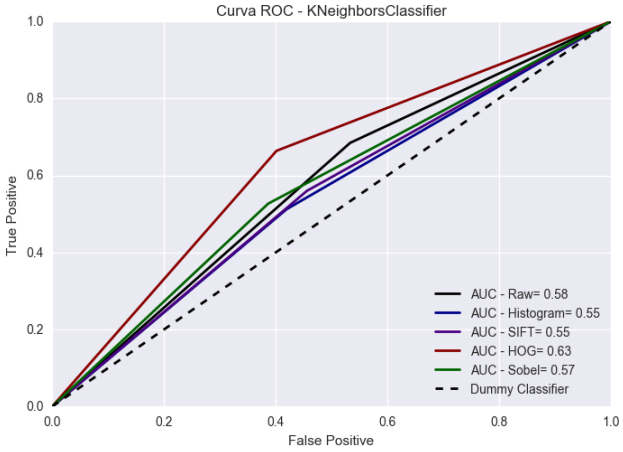
Se fosse o caso, as curvas ROC também poderiam ser mostradas e discutidas. Caso sim, precisaria incluir no texto acima a informação que essa forma de validação foi utilizada.



(a) Decision Trees



(b) Gaussian Naive Bayes



(c) k-NN

Figura 2: Curvas ROC.

Anexo 1 - função principal

As principais funções são listadas abaixo. Além da principal, tem-se:

- `createDescriptors()`: cria os descritores para cada imagem.
- `runClassif()`: roda os classificadores para cada descritor e conjunto ótimo de parâmetros

---

```
TRAIN_DIR = 'kaggle/train/'

ROWS = 128, COLS = 128, CHANNELS = 3, NIM = 500

train_dogs =   [TRAIN_DIR+i for i in os.listdir(TRAIN_DIR) if 'dog' in i]
train_cats =   [TRAIN_DIR+i for i in os.listdir(TRAIN_DIR) if 'cat' in i]

# slice datasets for memory efficiency: delete if using full dataset
train_images = train_dogs[:NIM] + train_cats[:NIM]
random.shuffle(train_images)

#Random search
rs = 1;

# create the label array
labels = []
for i in train_images:
    if 'dog' in i: labels.append(1)
    else: labels.append(0)

#create the descriptors #this part can be improved
descrip = ['Raw', 'Histogram', 'SIFT', 'HOG', 'Sobel']
fullDesc = createDescriptors(train_images)

#run each classifier for each descriptor
evaluation = []
for desc in range(0,len(descrip)):
    (X_train, X_test, y_train, y_test) = train_test_split(
        fullDesc[desc], labels, test_size=0.25)

    print ('\nDescriptor: ', descrip[desc])
    #choose the best classifiers
    if rs==1:
        dt=best_dt(5)
        knn = best_knn(17)
    else:
        dt = DecisionTreeClassifier()
        knn =  KNeighborsClassifier(n_neighbors=3)

    classifiers = [knn, dt]

    #evaluate
    evaluation.append(run_classifiers(X_train, X_test, y_train, y_test, classif

plot_roc(evaluation, classifiers, descrip, rs)
plot_acc(evaluation, classifiers, descrip, rs)
plot_f1(evaluation, classifiers, descrip, rs)
```

```
def createDescriptors(train_images):
    rawImages = []
    descHist = []
    descSift = []
    imageHog = []
    descSobel = []

    count = len(train_images)

    for i, image_file in enumerate(train_images):
        image = read_image(image_file)

        # raw
        image_gray = read_image(image_file, cv2.IMREAD_GRAYSCALE)
        pixels = image_to_feature_vector(image_gray)

        # histogram
        histogram = extract_color_histogram(image)

        # sift
        descsift = n_bigest_magnitude_descriptor(image_file, 50)

        # hog
        imhog = desc_hog(image_gray)

        # sobel
        imsobel  = filters.sobel(image_gray)
        sobel_pixels = image_to_feature_vector(imsobel)

        rawImages.append(pixels)
        descHist.append(histogram)
        descSift.append(descsift)
        imageHog.append(imhog)
        descSobel.append(sobel_pixels)

        fullDesc = [rawImages, descHist, descSift, imageHog, descSobel]

    return fullDesc
```

```
def runClassif(X_train, X_test, y_train, y_test, classifiers):
    acc=[]
    f1=[]

    for clf in classifiers:

        clf.fit(X_train, y_train)
        pred = clf.predict(X_test)

        acc.append(clf.score(X_test, y_test))
        f1.append(f1_score(y_test, pred, average='micro'))

        fullDesc = [acc, f1]
    return fullDesc
```

Anexo 2 - Parâmetros dos algoritmos de classificação

```
Descriptor:  Raw
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=13, p=2,
           weights='uniform')
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=8,
            max_features=None, max_leaf_nodes=10, min_impurity_split=1e-07,
            min_samples_leaf=13, min_samples_split=11,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')


Descriptor:  Histogram
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=11, p=2,
           weights='uniform')
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=11,
            max_features=None, max_leaf_nodes=11, min_impurity_split=1e-07,
            min_samples_leaf=11, min_samples_split=17,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')


Descriptor:  SIFT
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=13, p=2,
           weights='uniform')
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=16,
            max_features=None, max_leaf_nodes=6, min_impurity_split=1e-07,
            min_samples_leaf=10, min_samples_split=14,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')


Descriptor:  HOG
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=15, p=2,
           weights='uniform')
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=16,
            max_features=None, max_leaf_nodes=10, min_impurity_split=1e-07,
            min_samples_leaf=18, min_samples_split=7,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')


Descriptor:  Sobel
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=11, p=2,
           weights='uniform')
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=1,
            max_features=None, max_leaf_nodes=7, min_impurity_split=1e-07,
            min_samples_leaf=2, min_samples_split=19,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```