INF394 - Processamento Digital de Imagens

Trabalho Prático 3 - Versão Final

Prof. Marcos Henrique Fonseca Ribeiro

Junho/Julho de 2017

1 Observações Gerais Sobre o Trabalho

1. Formato de entrega: arquivo *.zip*, contendo os arquivos pedidos ao longo do roteiro.
2. Meio de entrega: PVANet
3. Equipe: mesma do primeiro trabalho
4. Valor da atividade: 50 pontos

2 Parte I

Nesta primeira parte, deve-se utilizar a estrutura básica de programa com interface gráfica disponibilizada no PVANet ou a linguagem/tecnologia que preferir para desenvolver um software com as funcionalidades especificadas nas seções a seguir, valendo as seguintes observações **importantes:**

- Caso não vá utilizar uma linguagem multiplataforma, como Python ou Java, disponibilize juntamente com os arquivos entregues as instruções para instalação, compilação ou qualquer outra que for necessária para que uma pessoa consiga instalar e executar o produto final em Windows, Linux ou Mac.
- **Está vetado** o uso de qualquer biblioteca, módulo pronto, framework ou coisa que o equivalha que já implemente os algoritmos e técnicas de Processamento Digital de Imagens propriamente dito como, por exemplo, *OpenCV*. Você deve implementar estes recursos.
- **Está liberado** o uso de qualquer biblioteca, módulo, framework etc. que não seja para a finalidade descrita acima. Por exemplo, esteja à vontade para utilizar bibliotecas para manipulação de interface gráfica, manipulação de arranjos, matemática, gerenciamento de arquivos no SO etc.
- A **exceção** à regra acima está no uso de soluções para a carga de uma imagem a partir de um arquivo para um objeto, estrutura de dados ou

afins; para a visualização de imagens e para o salvamento das mesmas. Sendo assim, por exemplo, pode-se utilizar o módulo Python PIL (*Python Image Library*) no trabalho para ler dados de arquivos, salvar arquivos, exibir imagens na tela. Porém não se poderá utilizar os recursos já presentes na mesma no que diz respeito ao processamento da imagem como, por exemplo, a conversão para uma imagem binária.

Por fim, apesar de não ser obrigatório, recomenda-se a utilização da estrutura software desenvolvida para o primeiro trabalho. Porém, para fins de avaliação, será considerada só a parte que disser respeito ao conteúdo da disciplina, isto é, processamento de imagens, e for referente ao conteúdo novo do trabalho. Para fins de simplicidade, envie o resultado final completo, incluindo o conteúdo do primeiro trabalho, se for o caso. A especificação do trabalho encontra-se na Seção 2.1.

2.1 Funcionalidades a serem Implementadas

As funcionalidades descritas aqui seguem uma sequência lógica didática. Esteja livre para enquadrá-las na organização de menus da interface do software da forma que bem desejar.

- Filtro de Butterworth, usando domínio da frequência (para o filtro, especificar o raio de suavização)
- Filtro de passa-baixa, usando domínio da frequência (para o filtro, especificar o raio de corte)
- Filtro de passa-alta, usando domínio da frequência (para o filtro, especificar o raio de corte)
- Filtro de passa-faixa, usando domínio da frequência (para o filtro, especificar os dois raios de corte)
- Limiarização de Otsu
- Limiarização Global Adaptativa
- Limiarização por Média Móvel
- Quantização para k níveis de cor usando cluster (ver exemplo de pseudo-binária disponibilizado no esqueleto original)

Para os filtros do domínio da frequência, pode-se utilizar os módulos disponíveis nas bibliotecas *scipy* ou *numpy*. Para as limiarizações, implemente as mesmas usando como base a versão em tons de cinza da imagem original. Sempre priorize o método de conversão da luminância (*luminosity*) para fazer a conversão para escala de cinza.

3 Parte II

Atenção: como esta segunda parte não é focada totalmente em implementação, **está liberado** o uso de qualquer biblioteca disponível que conseguir, incluindo

as de processamento de imagens, como *OpenCV*. No entanto, esta parte **deve** ser implementada em Python, uma vez que as dicas e padronizações feitas aqui levam em conta bibliotecas da linguagem.

Recomenda-se a instalação de uma biblioteca chamada *scikit-learn* (`http://scikit-learn.org/stable/`) a fim de facilitar as tarefas propostas. Esta biblioteca traz consigo implementações de algoritmos e ferramentas clássicas e bastante úteis para o contexto de aprendizado de máquina.

O grupo deve criar um classificador automático de imagens, conduzindo um experimento conforme descrito a seguir. Primeiramente, deve-se escolher um contexto de aplicação e um conjunto de imagens para servirem de dados de treinamento e testes. Selecione imagens para compor dois experimentos:

1. Base 1: Imagens de 3 ou tipos de animais diferentes. Por exemplo: cão, gato, peixe, ave. Priorize imagens cujo maior conteúdo seja o animal propriamente dito, com o mínimo possível de plano de fundo. Corte, usando um software como o GIMP, os animais de imagens com mais objetos e elementos, se achar necessário. Minimamente, obtenha cerca de 150 imagens.

2. Base 2: Imagens de personagens de histórias em quadrinhos, desenhos animados e afins. O importante é que os personagens tenham estilo artístico parecido. As mesmas observações a respeito de priorizar imagens contendo somente o personagem vale aqui.

Observações importantes:

- As imagens obtidas **não** precisam ser do mesmo tamanho
- Lide apenas com imagens coloridas. Os testes contemplarão análise de imagens em tons de cinza, mas obtidas via aplicação de filtros
- Algumas imagens podem não estar no padrão RGB, mas com outra codificação, como RGBA, indexado etc. Converta todas as imagens para o padrão RGB durante a criação da base. A biblioteca PIL, do Python, tem um método simples de realizar a conversão, o mesmo valendo para a biblioteca OpenCV.
- Separe cerca de 10% das imagens (após a padronização do formato RGB) em uma pasta separada. Este será o **conjunto de validação** final para cada experimento e suas figuras só serão processadas pelos algoritmos após terminado todo o processo de treinamento dos classificadores
- Crie algum esquema que facilite a rotulação automática das imagens, para associa-las às classes correspondentes. Uma sugestão é impor algum padrão de nomenclatura nos arquivos das imagens, contendo o nome da classe da mesma. Assim, pode-se facilmente aplicar uma função sobre o nome do arquivo que extrai deste o nome da classe correspondente àquela imagem. Por exemplo: *ddd-classe*.jpg, onde *ddd* seria um número inteiro contador de imagens, com três dígitos e *classe* seria o nome da classe daquela imagem. A bilbiotca *os*, do Python, pode auxiliar em tarefas que envolvam o sistema de arquivos do Sistema Operacional, como varrer pastas, analisar nomes de arquivos, verificar permissões etc.

O objetivo será encontrar o melhor classificador que a equipe conseguir para cada tipo de base de dados e comparar/analisar os resultados obtidos. Nas próximas seções são descritos os passos que devem ser executados para os experimentos, que essencialmente seguirão o modelo mostrado nas aulas da disciplina.

3.1 Passo 1: Filtragem, Geração de Imagens Sintéticas e Preparação da Matriz de Dados

3.1.1 Filtros

Testaremos o desempenho dos classificadores mediante à aplicação prévia de diferentes filtros (já implementados pelos alunos em etapas anteriores do projeto) sobre as imagens das bases de dados. Incluiremos aqui, como etapa de todos os filtros, o redimensionamento de todas as imagens para um tamanho padrão. Sugere-se utilizar o paradigma de Orientação a Objetos, a fim de modularizar melhor o programa e prover melhor reaproveitamento de código. Implemente os seguintes filtros descritos adiante.

Antes, porém, uma observação a respeito do tamanho padrão a ser utilizado nas imagens. O tamanho padrão deve corresponder à largura igual a menor das larguras entre todas as imagens da base de dados, e a altura idem, a menor dentre todas as alturas de imagens presentes. Isso poderá distorcer formas de objetos nas imagens originais, mas não há problemas em relação a isso, é até, de certa forma, desejável, pois não deixa de ser uma forma indireta de se adicionar ruídos nos dados, sempre úteis para se obter classificadores robustos. Sendo assim, temos a seguinte lista de filtros para implementação:

- Básico: apenas redimensiona a imagem para o tamanho padrão, sem qualquer processamento adicional.
- Cinza: converte a imagem para tons de cinza e aplica o redimensionamento.
- Limiar: converte a imagem para tons de cinza, executa a Limiarização Global Adaptativa e redimensiona a mesma.
- Mediana: aplica filtro de mediana sobre a imagem (fixe a máscara em $5x5$) e em seguida aplica o redimensionamento
- MedCinza: aplica o filtro de mediana (mesma máscara), converte para tons de cinza e redimensiona
- MedLimiar: aplica o filtro de mediana (mesma máscara), converte para tons de cinza, aplica limiarização (mesma anterior) e redimensiona o resultado

3.1.2 Geração de Imagens Sintéticas

Após a aplicação do filtro sobre as imagens, deve-se ampliar a base de dados obtida, gerando-se sinteticamente alguns arquivos adicionais à base de dados, com base já nas imagens filtradas. Estas imagens sintéticas nada mais são do que cópias de algumas das imagens da base de dados (já filtradas), acrescidas de alguma variabilidade, como por exemplo, a adição de ruído gaussiano.

Sendo assim, sugere-se a seguinte metodologia para geração de imagens sintéticas:

- Selecione aleatoriamente 30% das imagens da base de dados e crie cópias das mesmas com espelhamento horizontal. Mantenha estas cópias em memória principal, para que sejam acrescidas à matriz de dados do classificador
- Repita a instrução do item acima, porém com espelhamento vertical agora
- Gere cópias de imagens com aplicação de ruído gaussiano de leve intensidade em cerca de 30% da base de dados já resultante da aplicação dos itens anteriores. Mantenha todas as imagens em memória principal para a montagem da matriz de dados

3.1.3 Preparação da Matriz de Dados

Para que os classificadores funcionem adequadamente, necessitamos de uma matriz, a qual chamaremos de X, deste ponto em diante, e um vetor coluna ao qual chamaremos de Y. Considere que a base de dados resultante dos passos das seções 3.1.1 e 3.1.2 possua n imagens, com m pixels cada uma no total (já que estão todas redimensionadas para o mesmo tamanho).

A matriz X será de dimensões $n \times m$, onde cada linha corresponderá a uma imagem, que já foi devidamente linearizada para corresponder a uma única linha desta matriz. Vale lembrar que, portanto, $m = l \times c$, onde l é o número de linhas da imagem e c o número de colunas da mesma. Sendo assim, os dados de todas as imagens estarão representados nesta matriz, que por fim terá um tamanho razoável em memória.

O vetor coluna Y terá dimensão de tamanho n, que também corresponderá ao número de imagens presentes na base de dados, porém seu conteúdo não será numérico, mas conterá os nomes das classes associadas à cada imagem. Sendo assim, para se saber a classe ao qual a imagem $X[i]$ da matriz, correspondendo à i-ésima linha de X, basta que se acesse a i-ésima linha de Y, ou seja $Y[i]$.

Todos os modelos de classificador são capazes de lidar com matrizes e vetores de classes nestes formatos, daí a conveniência desta etapa de preparação. Porém, apesar de em tese os dados já estarem prontos para o aprendizado de máquina ao término desta etapa, há um fator de conveniência que deve ser levado em conta.

Mesmo em uma imagem bastante pequena, digamos por exemplo, de 80×95 pixels, ao ser linearizada, gera um número de colunas considerável para X ($80 \times 95 = 7600$ pixels), tornando a classificação um problema de alta dimensionalidade. Pois, ao se enxergar X como uma base de dados em formato tabular, teríamos um número muito grande de atributos para cada dado para serem analisados, aumentando bastante o custo computacional do processo todo.

Desta forma, é conveniente que se reduza a dimensionalidade do problema. Para isto, utilizaremos a técnica de Análise de Componentes Principais (*Principal Components Analysis* - PCA, em inglês). Conforme explicado em sala, esta técnica visa, abrindo mão da precisão na representação dos dados, capturar o que há de mais relevante na variação e capacidade de explicar os dados, gerando novas dimensões para os mesmos, levando-se em conta apenas os fatores principais, cuja quantidade é informada pelo usuário.

A biblioteca *scikit-learn*, mencionada anteriormente, provê, no módulo *decomposition*, uma classe chamada *PCA* que encapsula e realiza este processo de redução das dimensões. Ao se instanciar o objeto, informa-se um parâmetro *n_components*, o qual indica com quantas componentes principais deseja-se que a base de dados transformada fique.

Digamos que se deseje trabalhar com apenas 25 componentes principais (uma espécie de "pseudo-atributos" mais relevantes dos dados). Intancia-se o objeto com 25 como valor para o referido parâmetro e, em seguida, executa-se o método *fit_transform* sobre X para que se faça a redução, resultando em uma nova matriz $X'_{n\times 25}$ que, esta sim, será utilizada para treinar o classificador.

Para efeitos de simplificação, chamaremos deste ponto em diante X' apenas de X, tendo-se em conta que X é a matriz final, resultante dos passos descritos nesta seção e nas duas anteriores. Para o trabalho, padronize o número de componentes no valor aqui sugerido, isto é, 25. Pode-se, se o grupo desejar, realizar alguns testes com valores maiores e menores que este e ver o impacto desta redução no resultado final. Porém, tenha em mente que isto tomará mais tempo de execução do experimento, embora simples de implementar. Além disso, não se trata de um teste obrigatório, portanto, somente implemente se estiverem seguros de que isso não comprometerá o prazo de entrega do trabalho.

3.2 Passo 2: Preparação e Treinamento dos Modelos Classificadores

Esta etapa do experimento que vamos conduzir consistirá em treinar e comparar os resultados de diferentes algoritmos de classificação junto às bases de dados preparadas no passo anterior. Os algoritmos pertencem a uma famílias distintas de classificadores e cada um tem suas particularidades. Como o assunto de aprendizado de máquina não foi visto aprofundadamente em sala, não vamos esmiuçar muito os parâmetros de cada técnica, a não ser aqueles mais simples ou que foram explicados em sala. Os algoritmos são descritos nas seções a seguir.

3.2.1 Rede Neural *Multilayer Perceptron*

Trata-se do modelo mais conhecido de Redes Neurais (RN). No Python, na biblioteca *scikit-learn*, encontra-se disponível por meio da classe *MLPClassifier*, do módulo *neural_network* e possui diversos parâmetros de calibração. Vamos fixar um e variar também apenas um outro deles, para efeitos dos testes realizados.

O primeiro parâmetro, que vamos fixar, é aquele chamado *random_state*, que se trata de como será definida a semente de geração de números aleatórios dentro das escolhas da RN. O objetivo de se fixar o valor é controlar os resultados obtidos em execuções diferentes do algoritmo, permitindo que resultados sejam replicados posteriormente. Estabeleça este parâmetro com um número inteiro de sua escolha, como $random\_state = 3$.

O segundo parâmetro é o *hidden_layer_sizes*, cujo valor padrão é 100, mas que pode ser controlado. Ele indica o número de camadas internas do modelo *Perceptron*, conforme explicado em sala. Os testes do trabalho deverão contemplar os valores $10, 50$ e 100 camadas.

Sendo assim, algumas configurações distintas de classificadores terão de ser testadas: três configurações distintas de redes neurais para cada uma das possibilidades de filtros da base. Mais sobre os resultados e como comparar será

visto adiante. Antes de prosseguir, vale ressaltar que deste ponto em diante do texto vamos referenciar a este tipo de classificador apenas por **MLP**.

3.2.2 Máquina de Vetores Suporte para Classificação (SVC)

Trata-se de uma técnica de classificação também bastante utilizada na prática. Como mostrado em sala, consiste em obter o melhor ajuste possível para um tipo de curva de função que melhor separe (classifique) os dados em categorias distintas. No Python, na biblioteca *scikit-learn*, encontra-se disponível por meio da classe *SVC*, do módulo *SVM* e, como nos demais casos, possui diversos parâmetros de calibração. Neste caso, vamos fixar dois deles e variar outros dois.

O primeiro parâmetro que vamos fixar chama-se *random_state* e tem a mesma finalidade que apresenta no modelo MLP. Estabeleça este parâmetro com um número inteiro de sua escolha, como $random\_state = 3$. O segundo parâmetro que vamos fixar diz respeito ao número máximo de iterações que o algoritmo de ajuste de curvas pode executar. Dependendo da situação e das escolhas feitas, o algoritmo pode demorar muito a convergir e se tornar excessivamente custoso. Sendo assim, é conveniente estabelecer um limite máximo de testes. Estabeleça o parâmetro como $max\_iter = 500$.

Apesar de termos dois parâmetros a variar, um é dependente do outro e, na prática, funcionam mais ou menos como um só. São eles: *kernel* e *degree*. O primeiro, diz respeito ao tipo de função que será ajustada para funcionar como classificadora dos dados. Testaremos as opções: *linear*, *rbf* e *poly*, que estão entre os mais utilizados na prática.

RBF é uma sigla que vem de *radial basis function*. Uma função radial é aquela que tem a noção de centro (origem) dos dados e os valores da função dependem apenas da distância de cada ponto pra origem. Uma função gaussiana é um exemplo. Como o próprio nome sugere, o *kernel* linear é uma aproximação linear. Por fim, o *kernel* poly é uma aproximação polinomial. Somente quando se utiliza esta função é que deve-se fixar o valor do outro parâmetro mencionado, o *degree*, que vai estabelecer justamente o grau deste polinômio. Para os demais tipos de *kernel*, este parâmetro é simplesmente ignorado. Vamos fazer testes com os valores 3 e 5 para o grau do polinômio.

Sendo assim, temos quatro alternativas de funções aproximativas pra testar: linear, rbf, polinomial de grau 3 e polinomial de grau 5. Mais uma vez, cada configuração destas será usada com cada uma das possibilidades de filtros da base. Deste ponto em diante do texto vamos referenciar a este tipo de classificador apenas por **SVC**.

3.2.3 *Random Forest Classifier*

Em sala de aula, foi exemplificada uma técnica de classificação chamada Árvore de Decisão. O modelo *Random Forest Classifier* nada mais é do que uma espécie de "meta-classificador" que ajusta um certo número de árvores de decisão em várias sub amostras dos dados e utiliza informações médias de todas elas para ajustar suas previsões. Então, apesar de ser uma técnica mais elaborada e custosa, baseia-se numa técnica relativamente simples e amplamente conhecida, que foi ilustrada nas aulas. No Python, na biblioteca *scikit-learn*, encontra-se disponível por meio da classe *RandomForestClassifier*, do módulo *ensemble*

e, mais uma vez, possui diversos parâmetros de calibração. Também vamos trabalhar com apenas alguns deles, de forma parecida com os casos anteriores.

Novamente, o primeiro parâmetro que vamos fixar chama-se *random_state* e tem a mesma finalidade que apresenta nos demais modelos. Estabeleça este parâmetro com um número inteiro de sua escolha, como $random\_state = 3$. O segundo parâmetro que **podemos** fixar diz respeito à possibilidade de se treinar este modelo utilizando paralelismo. Como cada árvore do modelo é treinada de forma independente das demais, a implementação encontrada na biblioteca *scikit-learn* dá a possibilidade de que se treine árvores simultaneamente. Para isto, há um parâmetro chamado *n_jobs* no qual pode-se definir quantos processos simultaneos serão utilizados para o treinameto das árvores, agilizando o tempo total de resposta do treinamento do modelo. Se omitido ou definido com o valor 1, todo o processo será sequencial. Dependendo do seu hardware, a sugestão é que se estabeleça este parâmetro para o número de *cores* do processador do seu computador.

Um parâmetro númerico que deverá ser variado diz respeito ao número de árvores empregadas no modelo. Chama-se *n_estimators* e os valores testados devem ser 5, 10 e 20 árvores. O outro parâmetro que vamos variar é o *criterion*, no qual se define o método de cálculo do nível de qualidade das divisões do espaço de busca que cada decisão tomada na árvore impõe sobre os dados. Em sala de aula, foi mencionado que classicamente pode-se utilizar uma medida de entropia (similar àquela vista para calcular entropia em imagens). Porém, a implementação nos permite utilizar duas métricas: *entropy*, que corresponde à entropia propriamente dita, e *gini*, que tem a ver com o grau de impureza dos dados, mas para o qual não vamos entrar em detalhes aqui. Assim, além de variar o número de estimadores (árvores), deverá se variar o tipo de métrica, também, conforme mencionado aqui.

Mais uma vez, cada configuração destas será usada com cada uma das possibilidades de filtros da base. Deste ponto em diante do texto vamos referenciar a este tipo de classificador apenas por **RFC**.

3.2.4 K Vizinhos Mais Próximos (*K-nearest Neighbors*)

Nesta técnica, encontra-se entre os dados quais são os k indivíduos (instâncias, registros) mais parecidos com aquele que se deseja classificar e cada um deles "vota" na sua própria classe para que ela seja a escolhida como a classe do novo indivíduo. Por exemplo, digamos que num exemplo com $k = 10$, temos 3 vizinhos votando para que a classe do novo indivíduo x seja A, 1 vota para que a classe seja B, 6 votam para que a classe seja C e nenhum vota para que a classe seja D. Sendo assim, com alguma margem de erro, pode-se estimar a classe de x como sendo C, já que, entre os exemplos mais parecidos há uma predominância daquela classe, tornando esta a mais provável para x. Porém, comumente, pode-se querer estabelecer algum critério para se diferenciar os votos por peso, por exemplo, aumentando proporcionalmente o voto de um indivíduo o quão mais próximo ele está de x.

Feitas as explicações iniciais, assim como nos casos anteriores, há uma implementação disponível para este modelo na biblioteca *scikit-learn*. Procure pela classe *KNeighborsClassifier*, do módulo *neighbors*. Novamente, esta classe possui diversos parâmetros de calibração e também vamos trabalhar com apenas alguns deles.

Diferentemente dos casos anteriores, não há aleatoriedade nas decisões tomadas pelos algoritmos, então neste caso não há o parâmetro *random_state*. Porém, como no caso do RFC, há a possibilidade de que cálculos sejam feitos de forma paralela. Então há um parâmetro *n_jobs*, com o mesmo significado do caso anterior e que sugere-se a mesma coisa: definir o valor do mesmo para o número de *cores* do processador da máquina que realizará os testes.

Os dois parâmetros a serem variados são o número de vizinhos que vão participar da votação, denotado por *n_neighbors*, e o método de estabelecer o peso do voto destes vizinhos, denotado por *weights*. Para o primeiro parâmetro, testes devem ser feitos com 5, 10 e 20 vizinhos e para o método de atribuição de pesos, há duas alternativas: *uniform* e *distance*. O primeiro deles é o mais simples onde a distribuição de pesos é uniforme, isto é, todos os vizinhos votam com o mesmo peso. Na segunda alternativa, o peso do vizinho é inversamente proporcional à distância que ele está do indivíduo que está sendo classificado. Assim, além de variar o número de vizinhos, deverá se variar o tipo de distribuição dos pesos dos votos, conforme mencionado aqui. Novamente, cada configuração destas será usada com cada uma das possibilidades de filtros da base. Deste ponto em diante do texto vamos referenciar a este tipo de classificador apenas por **KNN**.

Para todos os quatro modelos apresentados, maiores detalhamentos de posibilidades de parâmetros, significado dos mesmos e até alguns exemplos de uso podem ser encontrados na documentação da biblioteca (em inglês). Veja o link para a mesma disponibilizado no início da Seção 3. A seção a seguir explica a metodologia que deve ser utilizada para o treinamento dos modelos.

3.2.5 Treinamento dos modelos

Em todos os casos, utilizaremos a metodologia de obtenção dos modelos a partir do esquema "treinamento, teste e validação". Para a validação final, uma parcela dos dados será isolada previamente, conforme instruído na Seção 3.1 e entraremos neste mérito mais adiante. Para as estapas de treinamento e teste, utilizaremos a técnica na qual se subdivide a base disponível em duas parcelas, uma onde o modelo será treinado e a outra onde a acurácia do modelo será testada e o mesmo pode ser ajustado em função disso.

Normalmente, define-se um percentual dos dados que será empregado no treinamento (a maior parte), e uma pequena parcela que será utilizada para se medir o desempenho. A escolha dos indivíduos que comporão cada uma das parcelas deve ser aleatória. Também, a fim de se avaliar com maior confiabilidade a acurácia do modelo de classificação, não se deve confiar no resultado obtido em um único procedimento destes de divisão da base, treinamento e teste, pois há a chance de que esta divisão favoreça um caso particular onde o modelo classificador tem bom desempenho, enquanto que se outras escolhas aleatórias fossem feitas, sua performance seria reduzida.

Sendo assim, uma prática recomendável é repetir este procedimento um certo número de vezes e se trabalhar com medidas estatísticas, como média e desvio padrão, do conjunto das avaliações feitas, de acordo com a(s) métrica(s) usada(s) para avaliar os resultados. Para auxiliar no processo de execução destas múltiplas avaliações, existem ferramentas na biblioteca *scikit-learn* que podem ser utilizadas para maior automatização. Porém, antes de detalhar este aspecto, uma observação a respeito de como avaliar os modelos classificadores deve ser feita.

Existem diversas métricas que podem ser utilizadas para se avaliar o desempenho de uma técnica de classificação. Pode-se, inclusive, como mostrado em aula, atribuir-se pesos distintos para diferentes tipos de erros cometidos. Literatura a respeito destas métricas pode ser facilmente encontrada em pesquisas na Web. Para este trabalho, no entanto, vamos adotar uma maneira mais simples. Avaliaremos apenas o chamado "score" de cada técnica, isto é, o percentual de acerto cometido por cada uma em suas avaliações.

Não serão feitas distinções entre os tipos de erros, como falsos negativos ou falsos positivos, e nem avaliaremos o score específico para a identificação de cada categoria. Em um curso específico sobre aprendizado de máquinas, estes detalhes e o processo de avaliação e comparação de técnicas seriam vistos com maior riqueza. Por aqui, vale só destacar que esta é uma etapa importante quando se quer desenvolver classificadores mais especializados em determinados tipos de contextos e situações. Por exemplo, se se deseja classificar objetos em n classes distintas, pode-se trabalhar com um esquema de um arranjo de n classificadores, cada um associado a cada uma das n classes e especializado em identificar se um objeto pertence ou não àquela classe em questão. Após isso, por um esquema de votação, por exemplo, pode-se utilizar um cálculo para estimar a classe mais provável. Neste tipo de contexto, o uso mais sofisticado de métricas seria mais importante.

Aqui, porém, visamos obter classificadores generalistas, o que, se por um lado simplifica o estudo e o desenvolvimento, por outro torna o problema mais difícil no que diz respeito à obtenção de resultados mais precisos. Tendo isso em mente, o trabalho não objetiva obter um classificador de alto desempenho na tarefa proposta, mas selecionar o melhor possível, dentre as possibilidades apresentadas aqui. O grupo deve estar ciente que esta tarefa, embora muito estudada e bem resolvida em uma grande quantidade de contextos atualmente, não é simples de se implementar e experimentação é necessária para tal. O principal intuito é colocar o aluno em contato com a área de análise inteligente de imagens, dentro do contexto de aprendizado de máquina.

Dando continuidade à descrição da metodologia, conforme mencionado, vamos utilizar dois recursos disponíveis na biblioteca de aprendizado de máquina para nos auxiliar nos testes. O primeiro deles é a classe *ShuffleSplit*, que auxiliará na divisão da base nas parcelas de treinamento e teste, enquanto o outro é a função *cross_val_score*, que executará os processos de treinamento e teste em cada uma das avaliações desejadas, computando as métricas de score e retornando as medidas em um vetor. Ambas, classe e função, estão presentes no módulo *model_selection*.

Tanto o contrutor da classe quanto a função aceitam diversos parâmetros. Vamos no ater apenas aos principais, para que consigamos conduzir o que está sendo proposto. Para construir o objeto *ShuffleSplit*, temos de definir a quantidade de vezes que faremos a divisão entre parcela de treinamento e teste, representado pelo parâmetro *n_splits*; o tamanho, isto é, o percentual dos dados que comporão a parcela de testes de uma divisão (um "split" - por exemplo: 0.2), representado pelo parâmetro *test_size* e, por fim, o parâmetro *random_state*, equivalente aos já vistos. Um experimento completo poderia trabalhar com variações também destes parâmetros, testando os algoritmos em diferentes esquemas e quantidades de "split". A equipe pode incluir isso no trabalho, se desejar. No entanto, não é necesário. Para fins de padronização, configure o objeto com 15 "splits", 15% da base para teste e semente aleatória arbitrária

qualquer.

Já para a função *cross_val_score*, temos dois parâmetros obrigatórios: *estimator*, que deverá receber um objeto classificador, previamente instanciado (no nosso caso, uma das diversas combinações dos modelos MLP, SVC, RFC ou KNN) e também o parâmetro X, para o qual se deve passar a matriz de dados. No nosso caso, a matriz **final** (após adição das instâncias sintéticas), obitda pela execução dos passos da Seção 3.1, é que será passada aqui.

Além destes dois obrigatórios, trabalharemos ainda com o parâmetro Y, para o qual passaremos o vetor de classes, também obtido nos passos da Seção 3.1; o *cv*, para o qual se passará o objeto *ShuffleSplit* instanciado anteriormente e, por fim, com o parâmetro *n_jobs*, equivalente aos já mencionados acima, sujeito às mesmas observações.

Importante: a função *cross_val_score* executa imediatamente ao ser evocada, e a mesma sozinha dispara o processo das múltiplas avaliações previstas no objeto *ShuffleSplit*. A execução desta função é o centro do experimento propriamente dito. Assim, ao terminar sua execução (que pode demorar ou não, dependendo de vários fatores como tamanho da matriz de dados, capacidade do hardware, dificuldade de uma técnica convergir para uma solução ótima etc), a mesma retorna um vetor numérico contendo o "score" de cada teste feito.

Por exemplo, se o parâmetro *n_splits* do objeto *ShuffleSplit* foi definido para 10 avaliações (chamaremos aqui de validações cruzadas), o resultado de *cross_val_score* será um arranjo de 10 valores númericos, pertencentes ao intervalo $[0, 1]$, representando o percentual de acerto do modelo de classificação para cada um dos "splits" (divisões) feitos na base. Este arranjo será a base para a métrica que vamos utilizar como medida de desempenho geral do modelo testado. Por "modelo testado", entenda-se cada combinação MLP, SVC, RFC e KNN gerada. Isto é, para cada combinação de calibragem pedida, uma bateria destas de validação cruzada deverá ser feita, gerando uma medida genérica de "score" para a mesma, descrita a seguir.

A maneira mais simples de se avaliar o desempenho geral de um conjunto de dados é a média. Porém, sabe-se que a média nem sempre pode ser uma bom jeito de se ter essa avaliação geral. Para isto, considera-se normalmente o desvio padrão também. Funções para cálculo automático de média e desvio padrão em qualquer estrutura de dados iterável (lista, tuplas, arranjos etc) podem ser encontradas na biblioteca *numpy*. Na nossa abordagem, vamos ser conservadores. Ao invés de se trabalhar simplesmente com uma média, vamos assumir, para cada teste realizado, a pior situação possível, isto é, dentro da variação central dos dados, o pior caso. Chamando-se nossa métrica de $F(m)$, onde m é uma referência o modelo classificador utilizado, temos:

$$F(m) = \bar{s} - \sigma_s, \tag{1}$$

onde, $\bar{s}$ é a média dos valores de "score" do arranjo referente à configuração de modelo classificador m e σ_s é o desvio padrão do mesmo conjunto de dados. Portanto, para selecionar o melhor classificador obtido, vamos considerar $F(m)$. Na próxima seção, vamos discutir a validação final dos modelos e como apresentar os resultados.

3.3 Passo 3: Seleção e Validação dos Melhores Classificadores

Nas etapas anteriores, preparamos a base de dados e treinamos e geramos métricas para avaliar classificadores automáticos de imagens. Vamos agora compará-los entre si e verificar como os mesmos se comportam mediante a dados novos, que ainda não foram submetidos aos algoritmos. Para isto, vamos utilizar aquela parcela de imagens que foi separada no início de todo o processo, à qual chamamos de dados de validação.

Antes, porém, vamos à seleção dos melhores classificadores. Crie uma tabela (para ser incluída no relatório) com informações que identifiquem cada configuração de modelo de classificação, como o tipo de algoritmo, os parâmetros utilizados e o filtro aplicado e associe a cada combinação a medida $F(m)$ obtida para o mesmo.

Se desejarem, para fins de melhor visualização da tabela e encurtar referência, pode-se criar um esquema de códigos para identificar cada calibragem de modelo. Se o fizerem, inclua como ler os código no relatório também. Montada a tabela, selecione, pelo critério de $F(m)$, os três melhores classificadores. Somente para estes classificadores será feita a etapa de validação, descrita a seguir.

Os modelos foram todos treinados com dados sujeitos a aplicação de filtros de imagens, linearização e redução de dimensionalidade. Portanto, antes de se perguntar ao classificador qual sua estimativa de classe para uma nova imagem, esta também precisa passar por este processo de tratamento. A abordagem que deve ser seguida é bastante próxima da original. Para cada um dos três melhores classificadores, siga os passos:

1. Crie e inicialize um contador de acertos para o classificador
2. Crie e inicialize um contador de imagens para o classificador
3. Para cada imagem da base de validação:
   (a) Usando o mesmo método de nomenclatura empregado anteriomente, extraia, do nome do arquivo da imagem, sua classe.
   (b) Aplique à imagem o filtro correspondente ao classificador selecionado.
   (c) Linearize a imagem (resultado: vetor)
   (d) Reduza a dimensionalidade do vetor da imagem. **Atenção**: este passo é sutilmente diferente daquele executado na preparação da base de treinamento e teste. Ver mais adiante.
   (e) Submeta a imagem processada ao classificador e obtenha sua estimativa de classe. **Atenção**: ver dica de como fazer mais adiante.
   (f) Compare a estimativa feita com a classe real da imagem (obtida no passo 1). Se iguais, incremente um contador de acertos para o classificador
   (g) Incremente o contador de imagens
4. Calcule o percentual de acertos do classificador na base de validação. Chamaremos esta medida de $V(m)$.

Para fazer a redução de dimensionalidade de uma imagem da base de validação, **não** se pode rodar um método de PCA novamente do zero. Por razões matemáticas que não serão detalhadas aqui, deve-se utilizar o mesmo ajuste dos dados utilizado originalmente. Portanto, deve-se manter salvo (em memória, ou persistido em disco) o objeto da classe *PCA* instanciado na Seção 3.1.3, para que a mesma transformação feita originalmente sobre os dados de treino seja aplicada aos dados da imagem de validação. Assim, usando o mesmo objeto instanciado para reduzir a dimensionalidade dos dados de treino, submeta o vetor da imagem de validação ao método *transform* do objeto PCA. Note que originalmente foi utilizado o método *fit_transform*, e agora, no mesmo objeto, deve ser utilizado o método *transform*. Não confundir estes passos. O método *transform* retornará um novo vetor correspondente à imagem, com o número de atributos reduzidos para a mesma quantidade da base de treinamento. Este será o novo vetor correspondente à imagem.

Para se obter a estimativa (predição) de um classificador para o vetor da imagem, basta evocar o método *predict* do mesmo objeto instanciado do classificador escolhido, passando como parâmetro o vetor da imagem. O retorno será o rótulo da classe que o classificador estimou para aquela imagem.

Já em preparação para o roteiro, monte uma tabela comparando as métricas $F(m)$ e $V(m)$ para cada um dos três classificadores eleitos para esta etapa do trabalho.

4 Entrega e Roteiro

4.1 Formato e Arquivos para Entrega

Enviar dois arquivos compactados *.zip*, nomeados *parte1.zip* e *parte2.zip*, referentes a cada parte do trabalho.

O **arquivo da Parte I** deve conter:

- Todos os arquivos *.py* (ou na linguagem escolhida) resultantes da versão final do programa desenvolvido pelo grupo.
- Arquivo de texto (*.pdf*) com um **relatório**, que deve:
  - Identificar o(s) aluno(s) que realizou(aram) o trabalho, com nome e matrícula
  - Conter instruções para instalação e execução do software desenvolvido como, por exemplo, quais pacotes necessitam estar instalados para o funcionamento, como compilar etc.
  - Breve explicação a respeito de cada item implementado, descrevendo, em linhas gerais, como o método foi traduzido para a linguagem em questão, quais otimizações foram feitas, quais simplificações (se for o caso) foram feitas, decisões de projeto tomadas etc. Lembre-se de incluir as fontes utilizadas, caso lide com algo que fuja ao material dado em sala.
  - Coloque também exemplos com imagens antes e depois de cada recurso implementado.

- Arquivos das imagens originais (porém em tamanho não muito grande) utilizadas pelo grupo para exemplificar os recursos no relatório, caso venham a utilizar imagens além das fornecidas pelo professor.

O **arquivo da Parte II** deve conter:

- Todos os arquivos *.py* com os scripts do experimento desenvolvido pelo grupo.
- Arquivo de texto (*.pdf*) com um **relatório**, que deve ter em seu cabeçalho identificação do(s) aluno(s) que realizou(aram) o trabalho, com nome e matrícula. O relatório deve seguir as instruções da próxima seção.
- Uma pequena amostra das imagens utilizadas no experimento, contendo cerca de 10 imagens de cada base de dados.

4.2 Relatório da Parte II

Basicamente, o relatório deve ser um texto explicando e contando a história do processo de experimento realizado pela equipe, tendo mais ou menos a seguinte estrutura geral de capítulos / seções:

1. Introdução. Deve dizer o objetivo do trabalho, explicar as escolhas de imagens, como foram obtidas e o porquê destas escolhas.
2. Montagem da base de dados. Explique **brevemente** como foi feita a preparação dos dados. Caso tenha adotado soluções diferentes das propostas aqui ou precise fazer observações sobre as decisões de projeto para geração da matriz de dados, é neste ponto que devem ser feitas.
3. Treinamento dos Modelos. Também explique **brevemente** como foram treinados os modelos. Assim, como no caso da seção anterior, este também é o local para observações que deseje fazer sobre o processo de treinamento.
4. Seleção e Validação dos Classificadores. Esta seção deve ter: a tabela comparando todos os modelos (em cada base de dados); a justificativa de escolha dos três melhores; explicação **resumida** do processo de validação e apresentação da tabela de resultados comparando $F(m)$ e $V(m)$ para os três melhores classificadores (em cada base).
5. Análise dos resultados e conclusões. Seguir roteiro a seguir.

O texto seção de análise de resultados e conclusões deve responder e justificar as respostas às seguintes perguntas:

1. Comparando a tabela final de resultados de $F(m)$ e $V(m)$, os modelos escolhidos tiveram a performance esperada? Por que?
2. Comparando a tabela final de resultados de $F(m)$ e $V(m)$, qual você apontaria como o melhor modelo? Por que?
3. Quais as principais dificuldades encontradas no desenvolvimento do trabalho?

4. O que você pode dizer da influência do filtro sobre os resultados obtidos? Justifique sua resposta com tabelas que, fixado um modelo e seus parâmetros, apresente a variação dos resultados mediante à variação dos filtros de imagem. Não precisa que se façam tabelas para todas as combinações, apenas para dois dos exemplos mais relevantes para cada base.

5. Analisando seus resultados para cada base de dados, para que casos e parâmetros você observou que houver maior sensibilidade dos resultados aos parâmetros (exceto o filtro)? Isto é, em que casos pode-se notar que a variação de um determinado parâmetro (exceto o filtro) influenciou mais na variação de qualidade das resposta?

6. Comparando agora cada contexto, a classificação de imagens de animais com a classificação de imagens de personagens desenhados, o que você pode tirar de conclusões a respeito do seu experimento? O que foi mais efetivo em cada caso e por que você **supõe** que isso tenha acontecido?

Bons estudos.
Prof. Marcos