

Relatório do Trabalho Final - Inteligência Computacional

Guilherme Almeida Lopes¹, Rafael Roseira Machado², Caio Rangel³

³Departamento Acadêmico de Ciência da Computação
Universidade Tecnológica Federal do Paraná - Campo Mourão

1. Dataset

O dataset utilizado para a realização do trabalho consiste em imagens que representam os meses do ano escritos à mão, contemplando uma variedade de formatos de escrita. O conjunto de dados é composto por um total de 4.200 imagens, das quais 80% foi destinada ao treinamento do modelo, enquanto a outra 20% foi reservada para teste do modelo. Mais especificamente, cada mês do ano possui 350 amostras de cada, totalizando 4200 imagens, sendo 840 testes e 3360 para o treinamento de cada modelo.



Figura 1: meses escritos à mão: abril, setembro e outubro.



Figura 2. meses escritos à mão: Novembro, maio e março.

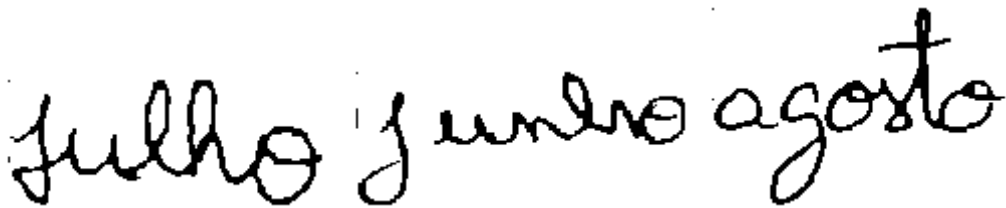


Figura 3. meses escritos à mão: julho, junho e agosto.



Figura 4. meses escritos à mão: fevereiro, janeiro.

2. Descritores de imagem

A definição e implementação de descritores de imagem são etapas essenciais no processo de extração de características, uma fase fundamental para a construção de sistemas de reconhecimento de padrões. Por meio dessa etapa, os pixels de uma imagem são transformados em vetores de características, permitindo que o modelo reconheça padrões de forma mais eficiente, em comparação ao uso de métodos convencionais baseados apenas em medidas diretas da imagem.

No presente trabalho, utilizamos descritores de textura, sendo particularmente eficazes para identificar padrões visuais e propriedades de superfícies, observados através das variações de intensidade dos pixels e das cores presentes na imagem. Os descritores adotados foram: **GLCM (Gray-Level Co-occurrence Matrix)**, **LBP (Local Binary Patterns)** e **LPQ (Local Phase Quantization)**. Cada um desses métodos contribui para a captura de diferentes aspectos texturais, enriquecendo o processo de extração de características e melhorando a capacidade do modelo em reconhecer os padrões.

2.1. Gray Level Co-occurrence Matrix (GLCM)

A matriz resultante da GLCM descreve a frequência de um determinado nível de intensidade de cinza em uma imagem, considerando não *pixels* isoladamente, mas uma vizinhança. Já que é necessário haver uma vizinhança bem definida, é preciso definir uma direção e uma distância para ser estabelecida a abrangência da vizinhança, como também a quantidade de níveis de intensidade de cinza. Em nosso código utilizamos os valores 1, 2, 4, 8, 16, 32, 64, 128 e 256 para distância e 0°, 45°, 90°, 180° e 270° para a direção. Uma vez efetuado o cômputo da matriz, utilizamos diversas métricas para poder derivar as características de textura da imagem: contraste; dissimilaridade; homogeneidade; ASM; energia; correlação.

As distâncias utilizadas visam capturar diferentes escalas de variação na textura, abrangendo tanto distâncias curtas quanto longas entre os pixels. No trabalho. Distâncias curtas, como 2, 4 e 8 pixels, permitem identificar texturas mais finas, destacando detalhes locais e características sutis, como o estilo de escrita do nome do mês. Por outro lado, distâncias longas, como 16, 32, 64 e 70 pixels, ajudam a capturar padrões texturais mais amplos, fornecendo informações em uma escala maior, como características gerais ou uniformidade na distribuição da textura. Essa abordagem multiescalar contribui para uma análise mais abrangente, permitindo que o modelo extraia tanto detalhes específicos quanto padrões globais da imagem.

2.2. Local Binary Patterns (LBP)

O Local Binary Patterns (LBP) é um algoritmo de descrição de textura utilizado no reconhecimento de padrões. Para cada pixel, seleciona-se um pixel central que é comparado com seus vizinhos em uma região de tamanho R(raio). Se o valor do pixel central for maior ou igual ao vizinho, atribui-se 1; caso contrário, atribui-se 0. Essa comparação gera uma sequência binária, que é convertida para um valor decimal, substituindo o valor do pixel central. O resultado é uma imagem transformada, onde cada pixel é representado por seu valor binário convertido.

Posteriormente, o histograma dos pixels é retornado como um vetor de características, que é utilizado no treinamento do modelo. Para esse processo, foram empregados os seguintes parâmetros: $P=25$, $R=6$ e o método **nri_uniform**. Essa técnica se mostrou extremamente útil para a identificação de padrões locais e, quando comparada a outros extratores de características, demonstrou ser altamente eficiente para o dataset utilizado.

2.3. Local Phase Quantization (LPQ)

O Local Phase Quantization é um método de extração de características eficiente e robusto para análise de texturas, amplamente utilizado em aplicações de visão computacional. Ele utiliza uma abordagem baseada na Transformada de Fourier de curto termo (STFT) para capturar variações locais de fase em uma imagem. Essa técnica é especialmente útil para cenários onde a robustez contra ruído, mudanças de iluminação e desfoque é essencial.

A configuração utilizada foi **winSize = 25** e **decorr = 1**, em que o LPQ opera com uma janela maior, permitindo capturar características texturais de regiões mais amplas da imagem. Além disso, a decoração espacial é ativada para reduzir redundâncias entre frequências, aumentando a discriminação das características extraídas.

3. Vetor de Características

Para cada descritor de textura é retornado um vetor de características; como utilizamos três descritores diferentes, decidimos unir estes três vetores em único vetor de características. Os vetores de características individuais de cada descritor encontram-se na pasta DadosExtraídos, contudo, por um problema de formatação do arquivo, foi decidido fazer o LPQ não ser escrito em um arquivo, visto que com o aumento de P , o arquivo ia se tornando muito grande.

4. Pré-Processamento

Durante a fase de pré-processamento dos dados, aplicamos quatro recursos essenciais antes de submetê-los ao treinamento, pois algoritmos como KNN e SVM apresentam melhor desempenho quando os dados possuem escalas semelhantes e também precisam saber tratar variáveis categóricas. Para a SVM utilizamos o **Standard Scaler**; o **PCA (Análise de Componentes Principais)**; o **Label Encoder**; e para o algoritmo KNN, utilizamos todos os recursos já mencionados e ainda aplicamos também o **MinMaxScaler**. Nas próximas subseções apresentaremos mais detalhes sobre cada um deles.

4.1. Label Encoder

O Label Encoder foi utilizado para tratar a variável categórica *mês do ano*. Este tipo de variável representa um determinado grupo ou categoria, assumindo um número fixo e limitado de valores possíveis.

Outra alternativa muito utilizada é o One Hot Encoder, porém, neste contexto, faz mais sentido a utilização do Label Encoder já que os meses, naturalmente, possuem uma

relação de ordem e este transformador é responsável por preservar esta ordem, além de evitar que seja criada uma nova coluna para cada categoria.

O funcionamento do Label Encoder é o seguinte: para cada categoria possível, é atribuído um valor no intervalo de $[0, n-1]$, sendo que n representa o número de possíveis categorias. Como temos 12 meses em um ano, após aplicarmos o transformador, teremos categorias numeradas de 0 a 11, sendo que 0 representa janeiro; 1 representa fevereiro; e assim por diante.

4.2. Standard Scaler

O Standard Scaler faz o processo de padronização dos dados de tal forma que, para cada característica do conjunto de dados, a média seja zero e o desvio padrão seja um. Dessa forma, a fórmula abaixo descreve o funcionamento do Standard Scaler:

$$x' = \frac{x - \mu}{\sigma}$$

Sendo que x' representa o novo valor da característica; x é o valor atual da característica; μ é a média dos valores da característica antes da padronização; σ é o desvio padrão da característica antes da padronização.

4.3. MinMaxScaler

O MinMaxScaler normaliza os valores de uma característica, colocando-os entre um valor de 0 e 1. A fórmula abaixo descreve o funcionamento do MinMaxScaler:

$$x' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Sendo que x' representa o novo valor da característica; X é o valor atual da característica; X_{min} é o menor valor da característica; X_{max} é o maior valor da característica.

4.4. PCA

O PCA tem como objetivo principal diminuir a dimensionalidade de um *dataset* preservando o máximo possível as informações.

5. Modelos

Os algoritmos tradicionais têm limitações na identificação e classificação de padrões em imagens. Por isso, utilizamos modelos de machine learning, que oferecem maior capacidade de reconhecimento. No entanto, essas técnicas não garantem total precisão na saída do programa. Para melhorar a confiabilidade, podemos adotar um conjunto de estratégias que possibilitam boas estimativas. Além disso, é importante considerar o problema do overfitting, que ocorre quando o modelo se ajusta excessivamente aos dados de treinamento, comprometendo seu desempenho com dados do mundo real.

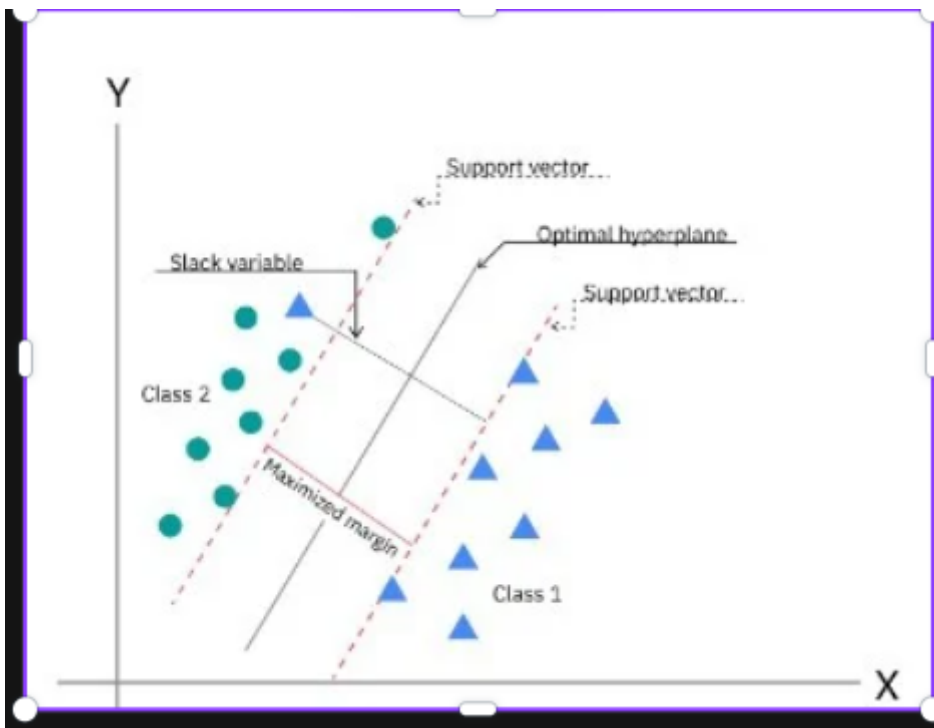
Para a realização deste trabalho, foram utilizados os algoritmos tradicionais K-Nearest Neighbors (KNN) e Support Vector Machine (SVM), comumente empregados em tarefas de classificação.

5.1. SVM

O SVM é um algoritmo de aprendizado supervisionado usado para classificação e regressão. Ele busca encontrar um hiperplano que melhor separa os dados em diferentes classes. Esse hiperplano é definido com base nos vetores de suporte, que são os pontos mais próximos da fronteira de decisão e determinam sua posição.

No caso de um problema linearmente separável, o SVM encontra o hiperplano que maximiza a margem entre as classes. Se os dados não forem linearmente separáveis, ele pode utilizar o truque do kernel, que transforma os dados para um espaço de maior dimensão onde eles possam ser separados linearmente. Alguns exemplos de funções de kernel incluem o kernel linear, polinomial e RBF.

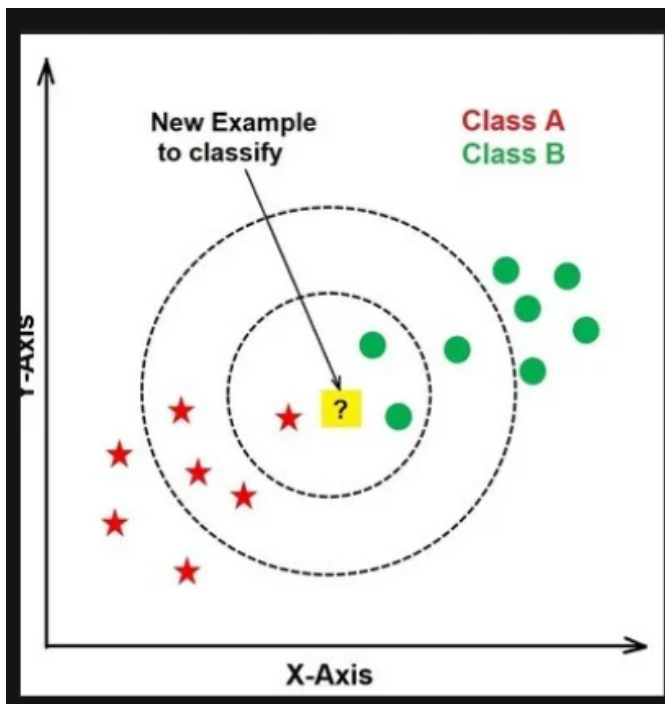
O SVM é muito eficiente em espaços de alta dimensão e funciona bem mesmo com um número limitado de amostras, ao focar somente nos vetores de suporte. Porém, pode ser computacionalmente custoso em grandes conjuntos de dados. Além disso, a escolha do kernel e dos hiperparâmetros, como o parâmetro C , que controla a penalidade de classificações erradas, influencia bastante seu desempenho.



5.2. KNN

O KNN (K-Nearest Neighbors) é um algoritmo de aprendizado supervisionado usado para classificação e regressão. Ele funciona com base na ideia de proximidade, classificando um novo ponto de dados com base nos seus vizinhos mais próximos. Para isso, calcula a distância entre o novo ponto e os demais pontos no conjunto de treinamento, geralmente usando métricas como a distância Euclidiana. O número de vizinhos considerados, representado por K , é um parâmetro essencial que influencia o desempenho do modelo. Se K for pequeno, o modelo pode ser muito sensível a ruídos nos dados, enquanto um K muito grande pode suavizar demais a fronteira de decisão, tornando o modelo menos preciso. No caso de classificação, o rótulo do novo ponto é determinado pela maioria dos rótulos dos K vizinhos mais próximos.

O KNN é simples e intuitivo, mas pode ser computacionalmente custoso para grandes conjuntos de dados, pois precisa calcular a distância entre o novo ponto e todos os pontos do conjunto de treinamento. Além disso, seu desempenho pode ser impactado pela escolha da métrica de distância e pela distribuição dos dados. Para melhorar sua eficiência, técnicas como KD-Trees e algoritmos de hashing podem ser usadas para otimizar a busca pelos vizinhos mais próximos.



5.3. RandomSearchCV

O RandomizedSearchCV é uma técnica utilizada para a otimização de hiperparâmetros em modelos de Machine Learning. Diferente do GridSearchCV, que avalia todas as combinações possíveis de hiperparâmetros em um determinado espaço, o RandomizedSearchCV amostra aleatoriamente um número fixo de combinações, o que

pode reduzir significativamente o tempo de busca e encontrar configurações próximas do ótimo de maneira mais eficiente.

O principal benefício dessa abordagem está na redução do custo computacional, tornando-a especialmente útil quando há inúmeros hiperparâmetros ou quando o ajuste fino do modelo precisa ser realizado em tempo hábil. O `RandomizedSearchCV` permite definir a quantidade de combinações a serem testadas, garantindo um equilíbrio entre a exploração do espaço de hiperparâmetros e o tempo de processamento.

Outro fator relevante é que essa técnica pode evitar o sobreajuste decorrente da exaustividade do `Grid Search`, pois explora regiões mais amplas do espaço de busca sem a necessidade de testar todas as possibilidades. Para sua implementação, é necessário definir o modelo de aprendizado de máquina, um dicionário com os hiperparâmetros e seus respectivos intervalos de valores, além da métrica de avaliação utilizada. Com isso, o algoritmo executa a busca e retorna a melhor configuração de parâmetros para o modelo.

5.3.1. GridSearchCV

O `GridSearchCV` é uma técnica amplamente utilizada para a otimização de hiperparâmetros em modelos de aprendizado de máquina. Ele realiza uma busca exaustiva sobre um espaço pré-definido de hiperparâmetros, avaliando todas as combinações possíveis dentro dos valores especificados. A principal vantagem dessa abordagem é garantir que a combinação ótima de parâmetros seja encontrada, já que todas as opções são testadas, proporcionando uma busca mais rigorosa pelo melhor desempenho do modelo.

A aplicação do `GridSearchCV` envolve a definição de um modelo de aprendizado de máquina, juntamente com um dicionário contendo os hiperparâmetros e seus respectivos valores ou intervalos. O algoritmo então executa uma validação cruzada para cada combinação de parâmetros, avaliando o desempenho do modelo com base em uma métrica escolhida, como acurácia ou F1-score. Isso permite que se identifique a configuração mais eficaz para o problema em questão.

Apesar de ser uma abordagem extremamente confiável, o `GridSearchCV` pode ser computacionalmente dispendiosa, especialmente quando o número de hiperparâmetros ou os valores possíveis para cada um deles são elevados. O número de combinações cresce de forma exponencial à medida que mais parâmetros ou valores são adicionados, o que pode tornar a busca muito lenta e consumir muitos recursos computacionais.

Portanto, embora o `GridSearchCV` seja ideal para cenários em que é necessário testar todas as combinações possíveis de hiperparâmetros e onde o tempo de computação não seja uma restrição, ele pode não ser a melhor opção em problemas com um grande número de parâmetros ou em contextos que exigem uma solução rápida. Nesse caso, técnicas alternativas, como o `RandomizedSearchCV`, podem ser mais apropriadas.

5.4. Diferenças KNN e SVM

Os algoritmos K-Nearest Neighbors (KNN) e Support Vector Machine (SVM) são amplamente utilizados para problemas de classificação, porém possuem diferenças fundamentais em sua abordagem e desempenho.

O KNN é um algoritmo baseado em instâncias que classifica um novo dado com base na proximidade de seus vizinhos mais próximos. Ele calcula a distância entre os pontos no espaço de características e atribui a classe mais comum entre os k vizinhos mais próximos. Uma das principais vantagens do KNN é sua simplicidade e facilidade de implementação, além de não necessitar de um treinamento formal. No entanto, sua principal desvantagem é o alto custo computacional durante a inferência, pois é necessário calcular a distância de um novo ponto em relação a todos os dados de treinamento. Além disso, seu desempenho pode ser afetado por dados ruidosos e pela escolha do valor de k .

Por outro lado, o SVM é um algoritmo de aprendizado baseado em margens que busca encontrar um hiperplano que melhor separa as classes no espaço de características. Ele utiliza vetores de suporte para definir a fronteira de decisão, maximizando a margem entre as classes. Se os dados não forem linearmente separáveis, o SVM pode utilizar funções de kernel para transformar os dados em um espaço de maior dimensão, onde a separação se torna possível. O SVM é mais eficiente em espaços de alta dimensionalidade e lida melhor com dados complexos. No entanto, seu treinamento pode ser mais demorado e a escolha do kernel e de hiperparâmetros adequados é crucial para um bom desempenho.

Em termos de desempenho no presente trabalho, o SVM obteve um F1-score de 77%, enquanto o KNN apresentou um desempenho inferior, com um F1-score de 52%. Isso demonstra que o SVM foi mais eficaz na classificação do conjunto de dados devido à sua capacidade de lidar melhor com alta dimensionalidade e padrões mais complexos. Enquanto o KNN pode ser útil em conjuntos menores e com padrões mais simples, o SVM se mostrou mais adequado para o problema estudado.

6. Métricas

Para avaliar o desempenho dos modelos KNN e SVM na tarefa de classificação das imagens, foram utilizadas diversas métricas amplamente empregadas em machine learning. Essas métricas permitem medir a eficácia dos modelos em diferentes aspectos e compreender seus pontos fortes e fracos.

6.1 Acurácia

A acurácia mede a proporção de previsões corretas em relação ao total de amostras. Embora seja uma métrica intuitiva, pode ser enganosa quando há

classes desbalanceadas, pois um modelo pode obter alta acurácia apenas classificando corretamente a classe majoritária.

A precisão (Precision) indica a proporção de verdadeiros positivos entre todas as previsões positivas feitas pelo modelo. Essa métrica é crucial quando o objetivo é minimizar falsos positivos e é definida pela fórmula:

$$Precision = \frac{TP}{TP + FP}$$

onde TP (True Positives) representa os casos corretamente classificados como positivos e FP (False Positives) são os casos incorretamente classificados como positivos.

6.2 Recall

O recall (revocação ou sensibilidade) mede a capacidade do modelo de identificar corretamente todas as instâncias da classe positiva, ou seja, a proporção de verdadeiros positivos em relação ao total de amostras que realmente pertencem à classe positiva. Essa métrica é essencial em problemas onde minimizar falsos negativos é mais importante. Sua fórmula é:

$$Recall = \frac{TP}{TP + FN}$$

onde FN (False Negatives) representa os casos positivos que foram classificados erroneamente como negativos.

6.2 F1-Score

O F1-score é a média harmônica entre precisão e recall, sendo especialmente útil quando há um equilíbrio entre a importância de minimizar falsos positivos e falsos negativos. Um F1-score alto indica que o modelo possui tanto uma boa precisão quanto um bom recall. A fórmula é:

$$F1score = \frac{2 * Recall * Precision}{Recall + Precision}$$

Além disso, a matriz de confusão é uma representação visual das previsões do modelo em relação aos rótulos reais. Cada célula da matriz indica a quantidade de previsões corretas e incorretas para cada classe, permitindo uma análise mais detalhada dos erros cometidos pelo modelo.

7. Implementação do Software

Para a realização deste trabalho, primeiramente definimos o dataset, sendo escolhida a base de dados "meses do ano", composta inicialmente por 6000 imagens. Posteriormente, parte das imagens foi retirada, especificamente as imagens entre as 351ª e 500ª de cada classe. Essa remoção ocorreu devido à falta de um servidor próprio para o software, o que limitou o processamento das imagens e do modelo.

Posteriormente, utilizamos o Joblib para paralelizar a extração de características das imagens como um todo e armazenar os vetores de características obtidos por cada extrator em arquivos. No entanto, o extrator LBP apresentou um problema ao adotar parâmetros na formatação do arquivo. Para resolver esse problema de forma simples, optamos por executar o LBP a cada vez que o software fosse executado, forma não ideal para desempenho do software como todo.

Após a extração de características, os vetores resultantes foram concatenados juntamente com o rótulo correspondente de cada imagem. Em seguida, os dados foram divididos em uma proporção de 80% para treinamento e 20% para testes, após a divisão do mesmo todos os rótulos dos casos de treinamento e teste foram retirados e armazenados em vetor de forma separada ambos, esta atitude foi tomada para garantir que quando a função train split embaralha-se os dados, os rótulos corretos fossem associadas as imagens corretas.

Os modelos escolhidos foram KNN e SVM, pois, a nosso ver, se encaixam melhor na extração de características para a natureza do problema apresentado. Inicialmente, utilizamos o GridSearchCV com um conjunto reduzido de parâmetros para otimizar o modelo, testando apenas os kernels: sigmoid, poly e rbf, além do parâmetro C nos valores 0.01, 0.1, 1, 10, 100 e 1000, garantindo no primeiro momento 60 % de f1-score, com validação cruzada de 5 partes.

Posteriormente, ao perceber que o modelo SVM ainda apresentava um desempenho abaixo do esperado, decidimos otimizar seus hiperparâmetros. Inicialmente, consideramos o uso do GridSearchCV, mas devido ao tempo excessivo que levaria para encontrar a melhor solução, optamos pelo RandomSearchCV. Essa abordagem permitiu realizar um número elevado de iterações sem aumentar significativamente o tempo de processamento. Os hiperparâmetros adotados para otimização foram os seguintes: para o PCA, ajustamos o número de componentes principais (n components) entre 16 e 2048 e o método de decomposição (svd solver) variando entre 'auto', 'full', 'arpack' e 'randomized'. Para o modelo SVM, ajustamos o parâmetro coef0 entre 0 e 20, o grau (degree) entre 1 e 20, e testamos os três tipos de kernel: 'poly', 'rbf' e 'sigmoid'. Além disso, o parâmetro C foi ajustado uniformemente logaritmo entre 10^{-5} e 10^3 , enquanto o parâmetro gamma foi testado com os valores 'scale' e 'auto'. O parâmetro de tolerância (tol) também foi ajustado logaritmo entre 10^{-5} e 10.

Além da otimização do modelo SVM, realizamos a busca por hiperparâmetros para o KNN utilizando o RandomSearchCV, permitindo explorar diferentes configurações sem um aumento significativo no tempo de processamento. Para a

redução da dimensionalidade, variamos o número de componentes principais (n_components) do PCA entre 1 e 32 e testamos os métodos de decomposição (svd_solver) 'auto', 'full' e 'randomized'. No modelo KNN, ajustamos o número de vizinhos (n_neighbors) entre 1 e 30 e testamos diferentes funções de ponderação dos pesos ('uniform' e 'distance'). A métrica de distância foi variada entre 'euclidean', 'manhattan' e 'chebyshev', enquanto o algoritmo de busca foi testado com as opções 'auto', 'ball_tree', 'kd_tree' e 'brute'. Além disso, variamos o tamanho da folha (leaf_size) entre 1 e 50 e o parâmetro p entre 1 e 2, ajustando a norma utilizada para o cálculo da distância.

8. Resultados

A seção a seguir apresenta os resultados obtidos pelos modelos KNN e SVM.

8.1 KNN Resultados

Os resultados obtidos foram extremamente positivos, embora ainda haja espaço para melhorias. O SVM, por ser um algoritmo capaz de lidar com alta dimensionalidade, apresentou um desempenho superior. Já no caso do KNN, foram realizadas diversas tentativas para reduzir a dimensionalidade da imagem, porém sem sucesso, pois transformar a descrição de uma imagem em um número reduzido de dimensões é um desafio extremamente complexo.

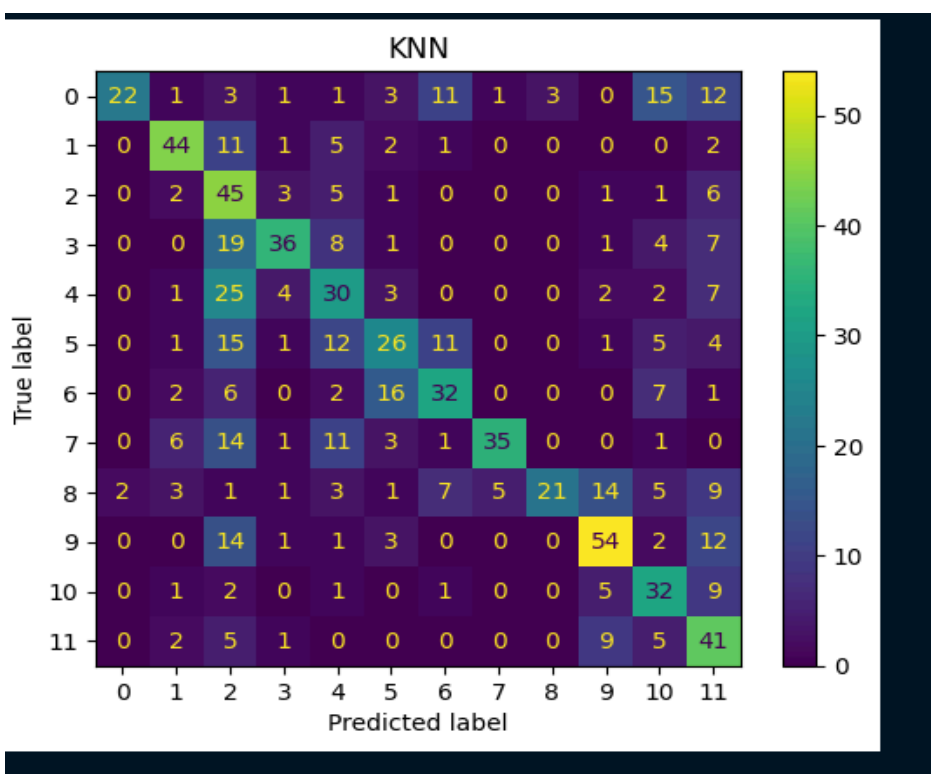
também foi considerada a possibilidade de reduzir cada componente da imagem para um único valor representativo no modelo KNN. No entanto, essa abordagem se mostrou desafiadora, pois o acúmulo de muitos pontos em um intervalo específico dificulta a discriminação adequada das classes, tornando o problema extremamente complexo, como pode ser visto na imagem a seguir.

O modelo Knn apresentou os seguintes resultados de cada métrica, para cada classe a seguir:

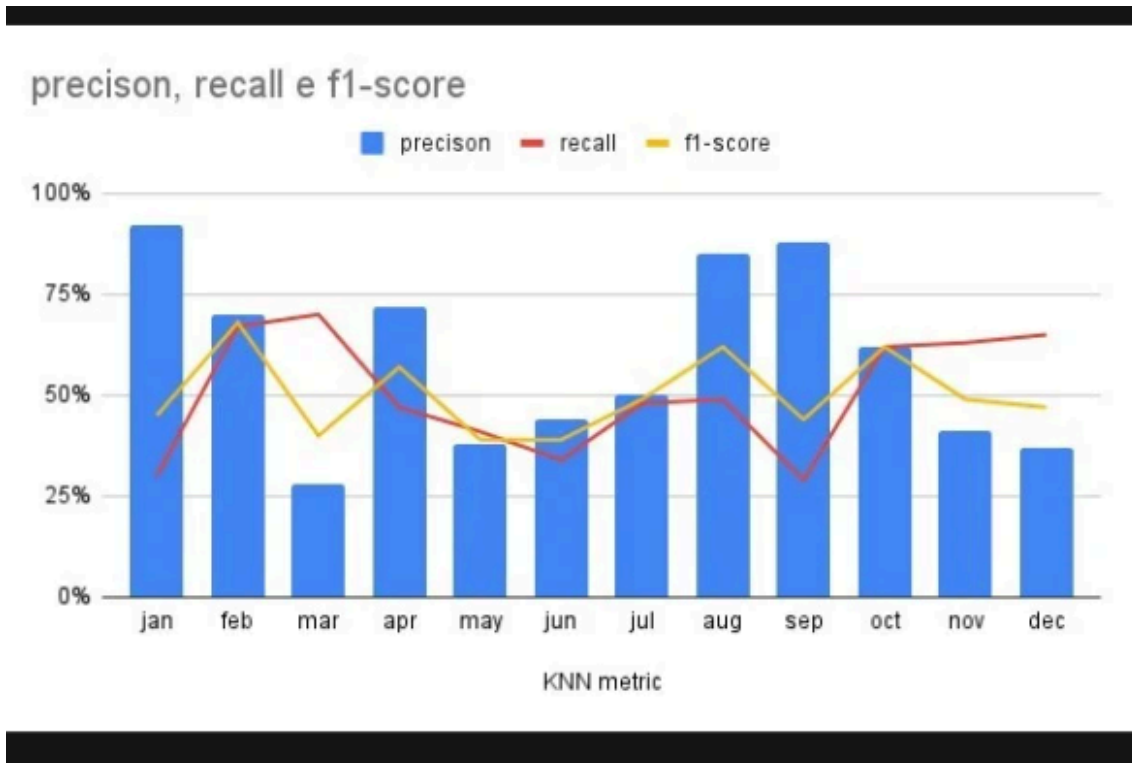
| Classe | Precision | Recall | F1-Score |
|--------|-----------|--------|----------|
| Jan | 92% | 30% | 45% |
| Feb | 70% | 67% | 68% |

| | | | |
|-------------|------------|------------|------------|
| Mar | 72% | 47% | 57% |
| Abr | 38% | 41% | 39% |
| May | 44% | 34% | 39% |
| Jun | 50% | 48% | 49% |
| Jul | 85% | 49% | 62% |
| Aug | 88% | 29% | 62% |
| Sept | 62% | 62% | 62% |
| Nov | 41% | 63% | 49% |
| Dec | 37% | 65% | 47% |
| | | | |

De modo geral, percebe-se que o KNN apresentou um F1-score de 50%, um valor considerado baixo. No entanto, ainda é possível obter uma boa estimativa da possível classe com base na previsão do modelo. Embora o resultado não seja ideal, ele pode ser extremamente importante e útil dependendo do contexto de aplicação, como pode ser visto na matriz de confusão a seguir.



A linha 9 é um exemplo a ser seguido. De modo geral, é possível determinar tendências nos resultados e nos erros das previsões do modelo, o que pode contribuir para a melhoria da sua capacidade de classificação.



O modelo KNN apresentou variação nas métricas ao longo dos meses. A precisão (barras azuis) teve meses de desempenho elevado, como janeiro, agosto e setembro, enquanto em outros foi mais baixa, mostrando uma distribuição irregular ao longo do tempo. O recall (linha vermelha) oscilou ao longo dos meses, mas manteve-se relativamente mais estável, sem variações extremas. O F1-score (linha amarela), que combina precisão e recall, seguiu um padrão semelhante, refletindo o impacto das oscilações nas outras métricas.

Em alguns períodos, a precisão foi alta enquanto o recall foi menor, e em outros ocorreu o oposto, o que influenciou diretamente o comportamento do F1-score. O desempenho do KNN não foi consistente ao longo do tempo, apresentando flutuações mensais nas métricas.



O gráfico apresentado exibe as métricas de desempenho de um modelo KNN, incluindo f1-score, recall e precisão. Observa-se que a métrica de f1-score (representada em azul) apresenta os valores mais altos nas categorias "macro avg" e "weighted avg", enquanto recall (vermelho) e precisão (amarelo) possuem valores semelhantes, porém inferiores ao f1-score. A acurácia geral do modelo é um pouco inferior ao f1-score médio, sugerindo que o desempenho do modelo pode variar entre classes individuais. De modo geral, os resultados indicam um desempenho equilibrado entre recall e precisão, sem grandes discrepâncias entre as métricas, o que é positivo para um classificador KNN.

8.2 SVM Resultados

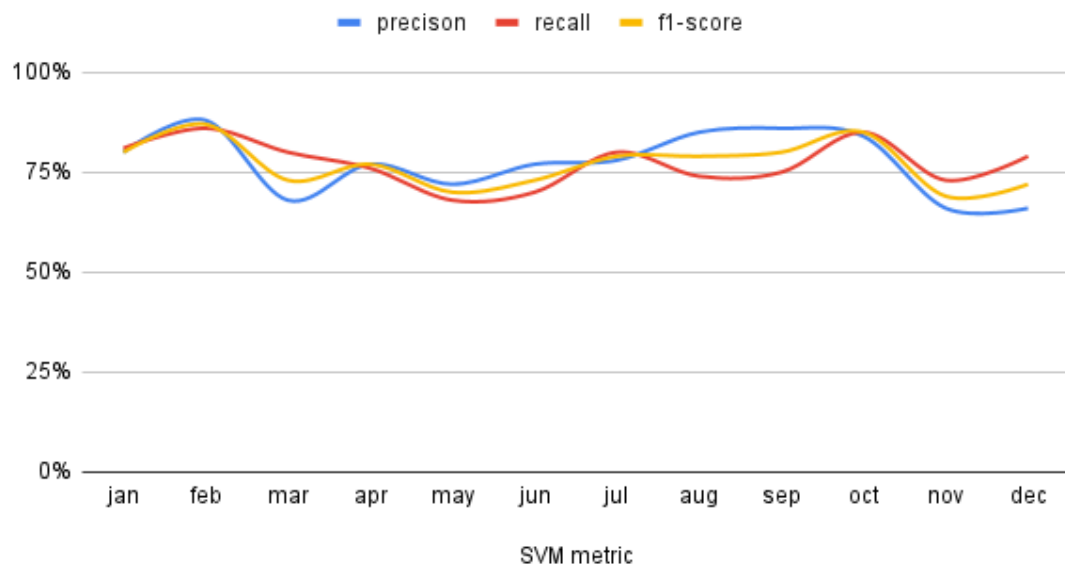
Os resultados indicam que o SVM apresentou um desempenho superior ao KNN, atingindo um F1-Score de 77%, o que demonstra uma maior eficiência na classificação. Além disso, as métricas de recall e precisão apresentaram valores muito próximos, evidenciando um equilíbrio entre a capacidade do modelo de identificar corretamente as classes positivas e minimizar falsos positivos. Esse comportamento sugere que o SVM pode ser uma escolha mais adequada para o problema em questão, garantindo uma melhor resolução global das classificações. A tabela a seguir ilustra esses resultados de forma mais detalhada, permitindo uma comparação direta entre os modelos.

| Classe | Precision | Recall | F1-Score |
|---------------|------------------|---------------|-----------------|
| Jan | 80% | 81% | 80% |
| Feb | 88% | 86% | 87% |
| Mar | 68% | 80% | 73% |
| Abr | 77% | 76% | 77% |
| May | 72% | 68% | 70% |
| Jun | 78% | 80% | 79% |
| Jul | 85% | 74% | 62% |
| Aug | 86% | 75% | 80% |
| Sept | 84% | 85% | 85% |
| Nov | 66% | 73% | 69% |
| Dec | 66% | 79% | 72% |
| | | | |

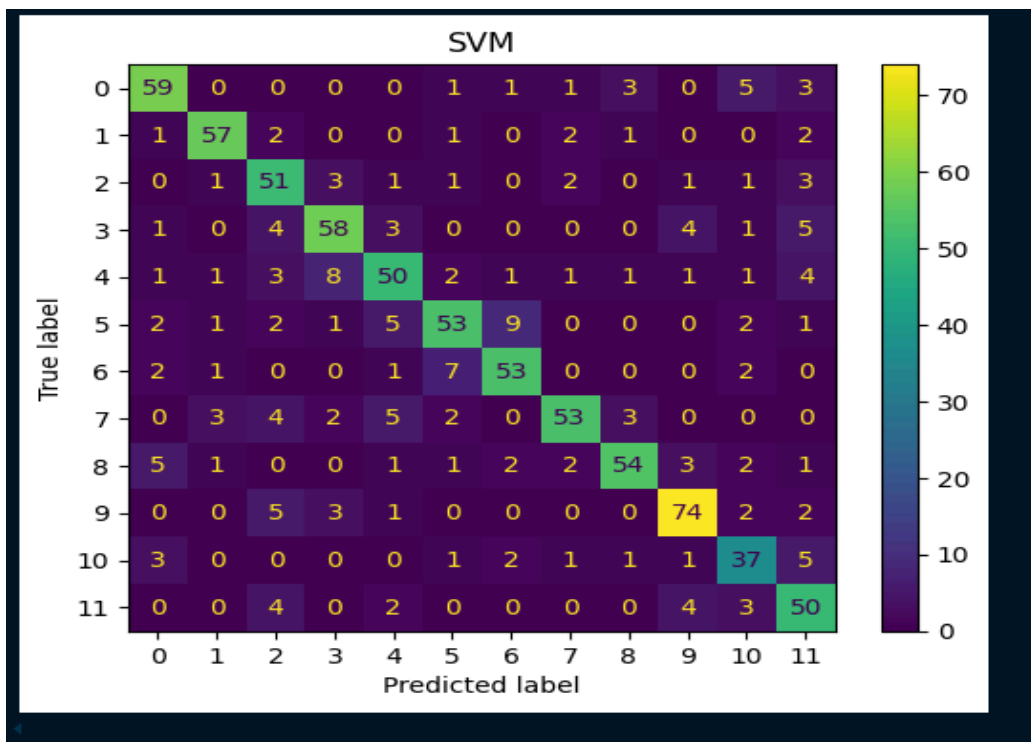
Conforme analisado na tabela acima, o modelo SVM demonstrou uma capacidade significativamente maior de identificar verdadeiros positivos e falsos negativos com precisão. Dessa forma, ele apresenta uma alta confiabilidade em suas predições.

A parte interessante que como pode ser visto no gráfico a seguir, que a taxa de recall e precision, se mostrou próximas garantindo que f1-score. Quando o recall e a precisão estão altos, o modelo se destaca em duas áreas cruciais. A alta precisão significa que a maioria das previsões positivas é realmente positiva, evitando falsos positivos. O alto recall indica que o modelo está identificando a maior parte das instâncias positivas reais, reduzindo os falsos negativos. Quando ambos estão elevados, o modelo alcança um bom equilíbrio, refletido em um F1-score alto. Essa métrica combina precisão e recall em um único valor, fornecendo uma avaliação mais robusta do desempenho do modelo, especialmente quando nenhum tipo de erro deve ser mais tolerado que o outro.

precision, recall e f1-score



E o mesmo pode ser observado na matriz de confusão a seguir:



9. Conclusão

O presente trabalho obteve um F1-score de 77% para o SVM e 52% para o KNN. O SVM apresentou melhores resultados em comparação ao KNN, devido à sua capacidade de lidar com a alta dimensionalidade dos dados obtidos pelo extrator de características. Acredito que a adoção de outras técnicas de padronização dos dados e transformações poderiam levar a um aumento substancial no F1-score e na acurácia do modelo KNN. No entanto, devido à natureza do problema, com muitas classes e alta dimensionalidade, a utilização de uma Rede Neural Artificial (RNA) parece ser mais interessante para alcançar melhores resultados. Diferentemente do KNN, acredito não alcançar um desempenho satisfatório a ponto de ser aplicado no mundo real, que devido a questão tempo não foi capaz de ser implementada. Futuros trabalhos poderiam focar na implementação de uma RNA para explorar seu potencial de melhorar os resultados.

References

JOBLIB.Parallel: joblib.em: <https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html>. Acesso em: 27/12/2024.

SCIKIT-LEARN. Machine learning in Python. Disponível em: <https://scikit-learn.org/stable/>. Acesso em: 15/01/2025.

STACK OVERFLOW. Stack Overflow – perguntas e respostas para programadores. Disponível em: <https://stackoverflow.com/questions>. Acesso em: 10/01/2025.