Frederick Salvador Tavares Prates

Renato Leite Risoli

Relatório Machine Learing I – Base Letter

1. Introdução

A base de dados analisada neste relatório foi produzida por SLATE,D. et al(1991) para criar modelos para identificar letras maiúsculas em formato de imagem. Em seu artigo, SLATE avaliou modelos como o KNN (K-nearest neighbors), *Bayes* e *Back Propagation*.
As imagens das letras foram avaliadas em 16 atributos numéricos que variam num intervalo de 0 até 15 em seu valor.

Para enriquecer a base, foram adicionadas imagens distorcidas das letras acumulando um total de 20.000 amostras.
Nosso objetivo é gerar modelos de algoritmos vistos em aula, comparar suas métricas e desempenho utilizando o pacote Sci Kit learn.

2. Algoritmos de Aprendizagem de Máquina

Segue abaixo os algoritmos utilizados para a avaliação. Optamos por não utilizar os algoritmos de Regressão Linear, pois como serão avaliados 16 atributos, e o modelo não seria indicado para esse caso, pois dificilmente conseguiria gerar uma relação linear entre 16 atributos distintos. Outro modelo que decidimos não avaliar, foi o Perceptron isolado, pois um modelo semelhante, o MLP, também foi avaliado.

2.1 KNN (K-nearest neighbors)

É um algoritmo de aprendizado supervisionado que avalia a distância para fazer a clusterização de elementos em grupos e assim fazer a sua classificação.

2.2 Árvore de Decisão

É um algoritmo que de aprendizado supervisionado que busca encontrar regras para chegar à uma classificação.

2.3 Floresta Randômica

Recebe o nome de floresta, pois utiliza várias Árvores de Decisão

2.4 Regressão Logística

É um algoritmo estatístico que com base em probabilidade tenta classificar uma instância.

2.5 Naive Bayes

Também é um classificador probabilístico como a Regressão Logística, mas que avalia a probabilidade de classificação com base na distribuição do conjunto.

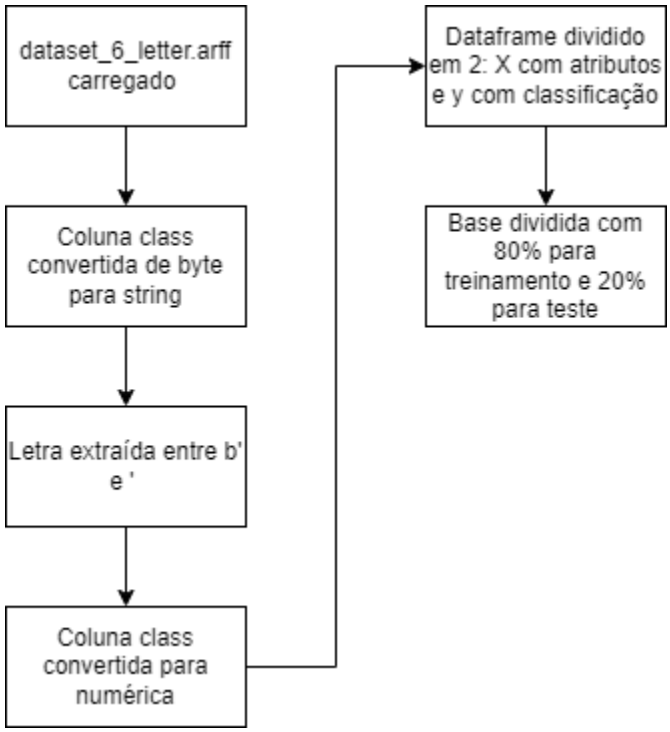
2.6 Perceptron e MLP

Utilizam o princípio das redes neurais. Um perceptron segue o modelo de um neurônio onde as entradas são os atributos a serem avaliados juntamente com um bias de ponderação. Esses atributos passam pela função de ativação e então a saída é a classificação. O MLP utiliza várias camadas de perceptrons para refinar o resultado.

2.7 SVM (Support Vector Machine)

Algoritmo que classifica os dados encontrando uma linha ou um hiperplano ideal que maximiza a distância entre cada classe em um espaço N-dimensional.

3. Experimentos



Utlizamos o Jupyter Notebook para analisar a base, juntamente com os pacotes Scikit learn para python que foi utilizado para gerar os modelos e métricas, e o pacote scipy que foi utilizado para ler o dado no formato ARFF em conjunto com o Pandas.

A base foi carregada em um Dataframe. Após transformações para facilitar a visualização das classes, o Dataframe foi dividido a partir de suas colunas, separando as

colunas de atributos, da coluna de classificação. Em seguida os Dataframes gerados, foram divididos em linhas, com 80% das linhas destinadas ao treinamento do modelo e 20% para testar o modelo gerado.

```python
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=199)
```

Foi definido também um K-fold estratificado de 10 camadas para a validação cruzada atribuído ao grid search.

Grid Search

```python
model = MLPClassifier()

parameters = {'learning_rate': ['invscaling', 'constant', 'adaptive'],
              'activation':['identity', 'logistic', 'tanh', 'relu'],
              'hidden_layer_sizes':[(16,26)]
              }

grid = GridSearchCV(estimator = model,             #
                    param_grid = parameters,       # dicionário com valores para serem testados (Pares Chave-Valor)
                    scoring = 'accuracy',          # métrica de avaliação
                    cv = kf)                        # cross-validation

grid.fit(X_train, y_train)

y_pred = grid.predict(X_test)

print("Melhores parâmetros:", grid.best_params_)
print("Melhor acurácia média:", grid.best_score_)
# performance no dataset de teste
print(classification_report(y_test, grid.predict(X_test)))
```

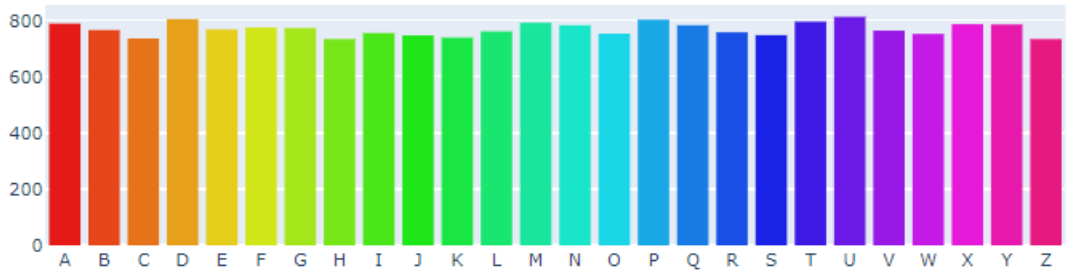
O grid search foi utilizado para encontrar os melhores parâmetros dentro de alguns selecionados para cada modelo de aprendizado de máquina. A quantidade de parâmetros avaliados foi limitada pela capacidade computacional disponível.

Ele recebe um modelo e os parâmetros desse modelo no formato dicionário para realizar combinações entre eles e avaliar o melhor desempenho, no caso deste relatório, por meio da acurácia.

3.1 Banco de Dados

O banco possui 26 categorias de letras que ficam na coluna “class”. Além da coluna target “class”, há também 16 colunas de features, que foram utilizadas para gerar os modelos classificadores a serem avaliados no relatório.

A base é bem balanceada como pode ser visto no gráfico abaixo:



Também não há a ocorrência de valores faltantes em nenhuma das 17 colunas.

Variable Name	Type	Description	Units	Missing Values
class	Categorical	capital letter		no
x-box	Integer	horizontal position of box		no
y-box	Integer	vertical position of box		no
width	Integer	width of box		no
high	Integer	height of box		no
onpix	Integer	total # on pixels		no
x-bar	Integer	mean x of on pixels in box		no
y-bar	Integer	mean y of on pixels in box		no
x2bar	Integer	mean x variance		no
y2bar	Integer	mean y variance		no
xybar	Integer	mean x y correlation		no
x2ybr	Integer	mean of x * x * y		no
xy2br	Integer	mean of x * y * y		no

x-ege	Integer	mean edge count left to right	no
xegvy	Integer	correlation of x-ege with y	no
y-ege	Integer	mean edge count bottom to top	no
yegvx	Integer	correlation of y-ege with x	no

A tabela acima descreve cada coluna da base com seu tipo, significado e se possui valores faltantes. Em todas as colunas do tipo inteiro, o range dos valores vai de 0 até 15.

3.2 Métricas

As 3 principais métricas utilizadas foram r2-score, erro médio quadrático e acurácia. Também foram geradas matrizes de confusão para avaliar as classificações.

3.2.1 R2-score

É pronunciado como R ao quadrado e é conhecido como coeficiente de determinação. Ele funciona medindo a variação nas previsões e o resultado da base de teste.

3.2.2 Erro médio quadrático

É comumente usados para verificar a acurácia de modelos e dá um maior peso aos maiores erros, já que, ao ser calculado, cada erro é elevado ao quadrado individualmente e, após isso, a média desses erros quadráticos é calculada.

3.2.3 Acurácia

É a taxa de acerto do modelo gerado em relação ao todo.

3.2.4 Matriz de Confusão

É uma tabela que permite a visualização do desempenho de um modelo classificatório. Ela apresenta de forma detalhada o resultado da classificação, comparando as previsões do modelo com os valores reais dos dados.

3.3 Resultados

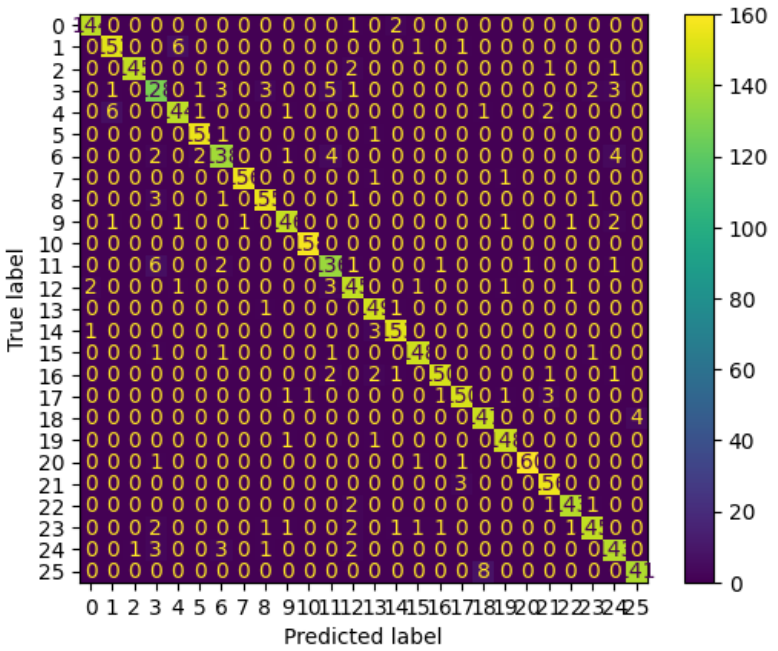
	R2 score	ESM	Acurácia
KNN	0.925	4.184	0.959
Árvore de Decisão	0.750	13.965	0.884
Floresta Randômica	0.871	7.206	0.935
Regressão Logística	0.503	27.794	0.774
Gaussian Naive Bayes	0.267	40.955	0.645
MLP	0.711	16.181	0.861
SVM	0.943	3.157	0.968

Os resultados acima mostram que o modelo gerado pelo algoritmo SVM foi aquele que desempenhou melhor, enquanto o modelo Naive Bayes Gaussiano desempenhou pior em relação aos outros avaliados.

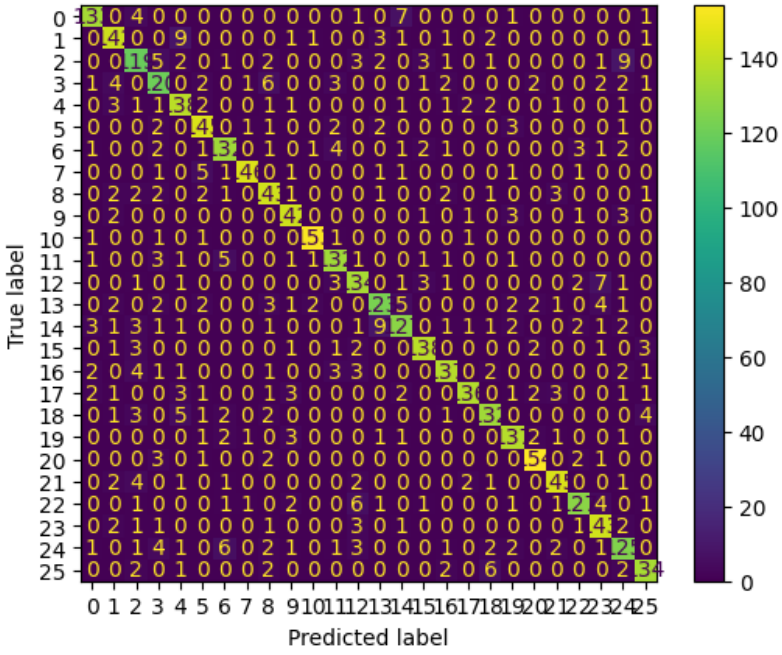
3.3.1 Matriz de Confusão geradas

Abaixo estão listadas as matrizes de confusão de cada modelo avaliado. É possível notar que em quase todos o eixo onde True Label = Predicted Label se destaca em relação aos erros de classificação. O modelo Naive Bayes Gaussiano, que teve o pior desempenho em termos de acurácia, gerou uma matriz, como esperado, com mais classificações erradas.

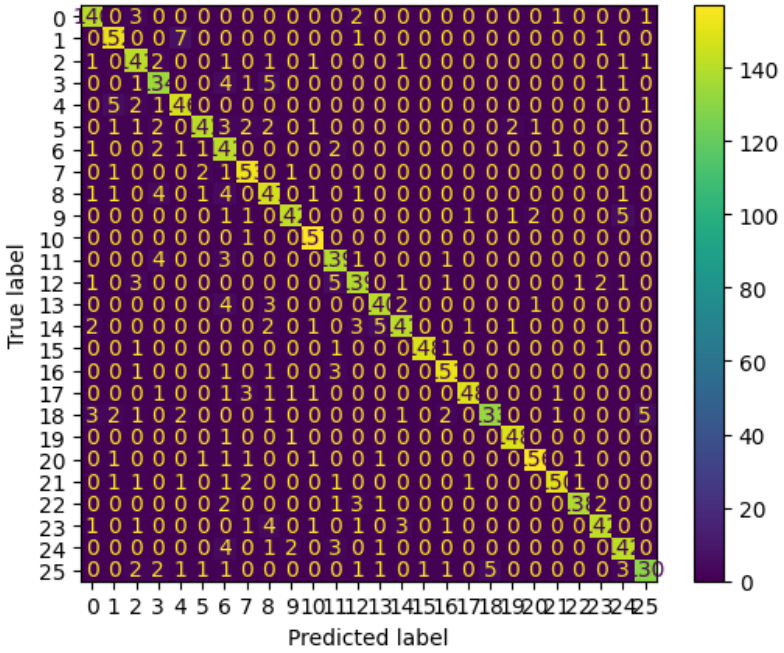
KNN



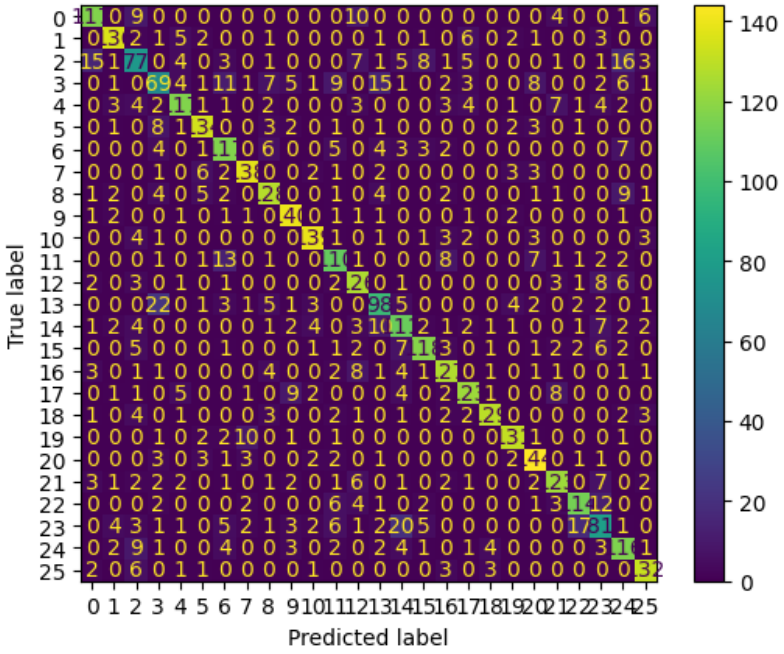
Árvore de Decisão



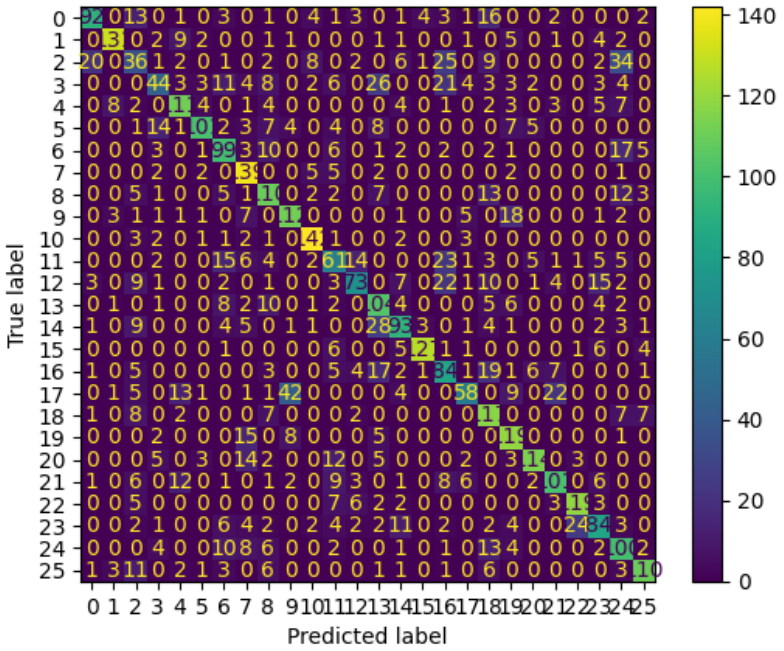
Floresta Randômica



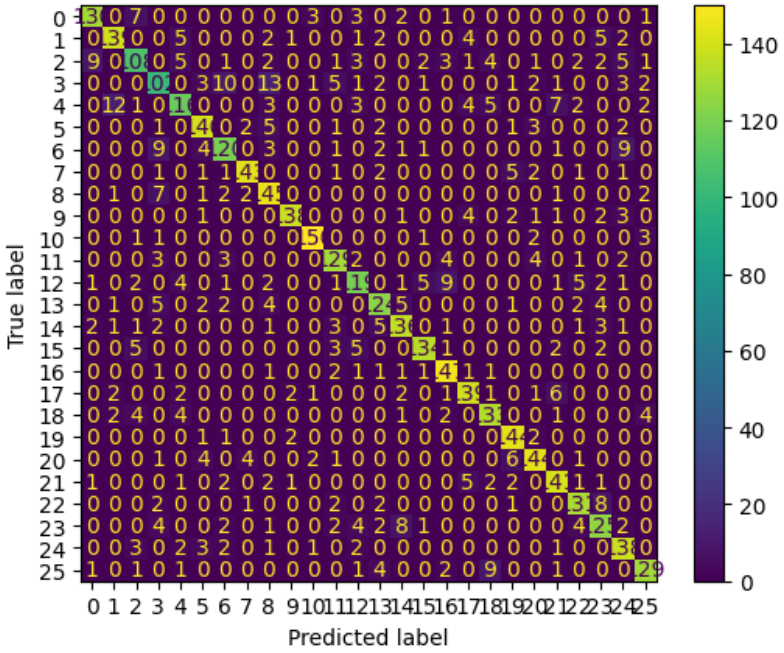
Regressão Logística



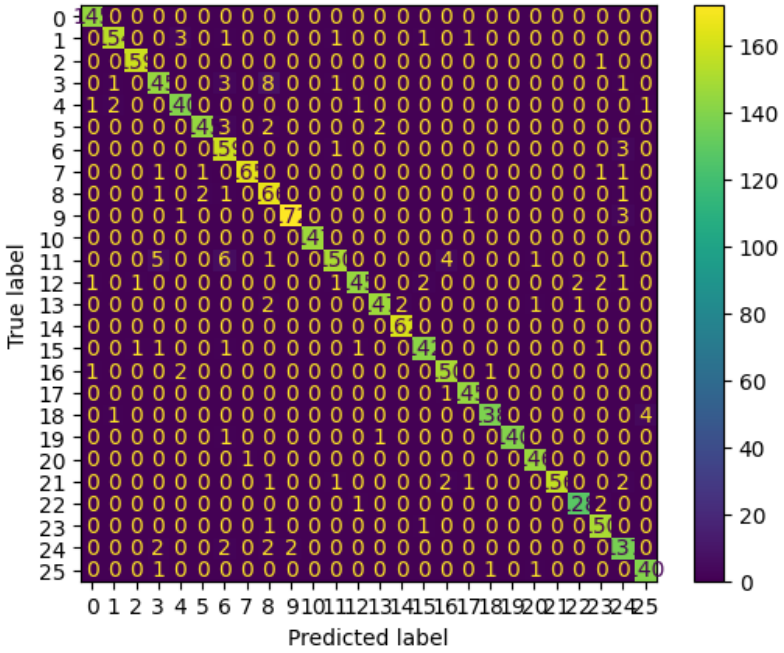
Gaussian Naive Bayes



MLP



SVM



3.3.2 Melhores parâmetros pelo Grid-search

KNN

```
model = KNeighborsClassifier()

parameters = {'n_neighbors':[1,100,1000,10000],
              'metric':["euclidean", "manhattan", "minkowski", "cosine", "l1", "l2"]
              }
```

Foram testados os parâmetros n_neighbors que varia a quantidade de vizinhos avaliados e as distâncias euclidiana, Manhattam, Minkowski, Cosseno, L1 e L2.

Os melhores parâmetros foram distância euclidiana e n_neightbors = 1.

Os parâmetros de números de vizinhos e métrica de distância foram selecionados, pois supomos serem os parâmetros que mais influenciariam no desempenho do modelo baseado na premissa de avaliação de vizinhos.

Árvore de Decisão

```
model = DecisionTreeClassifier()

parameters = {'criterion' : ['gini', 'entropy', 'log_loss'],
              'splitter':['best', 'random'],
              'max_depth':[1, 10, 100, 1000]
              }
```

Foram avaliados os parâmetros criterion do tipo gini, entropy e log_loss, o tipo de divisor best e randômico, e a profundidade variando de 1 até 1000.

Os melhores resultados foram obtidos com os parâmetros log_loss, profundidade 100 e divisor best.

Os parâmetros criterion e profundidade máxima foram selecionados, por maior familiaridade, já que foram parâmetros apresentados em aula. O parâmetro do divisor (splitter) despertou interesse, pois disponibilizava uma opção chamada 'best' e queriamos avaliar se realmente gerava uma divisão de nó melhor que a randômica.

Floresta Randômica

```
model = RandomForestClassifier()

parameters = {'criterion' : ['gini', 'entropy', 'log_loss'],
              'n_estimators':[1, 10],
              'max_depth':[1, 10, 100, 1000]
              }
```

Foram avaliados os parâmetros criterion do tipo gini, entropy e log_loss. Os parâmetros de quantidade de árvores (n_estimator) variando de 1 ou 10 e profundidade variando de 1 até 1000.

Os melhores resultados foram obtidos com 'criterion': 'entropy', 'max_depth': 1000, 'n_estimators': 10.

Os parâmetros criterion e profundidade máxima foram selecionados, por maior familiaridade, já que foram parâmetros apresentados em aula. O número de árvores na floresta gerou interesse, pois suspeitávamos que uma maior quantidade poderia servir para refinar o classificador.

Regressão Logística

```
model = LogisticRegression()

parameters = {'penalty' : ['l2'],
              'solver':['lbfgs', 'newton-cg', 'saga'],
              'C':[0.5, 1]
              }
```

Foram avaliados os parâmetros solver dos tipos lbfgs, newton-cg e saga que são compatíveis com a penalidade L2. O parâmetro C que é semelhante ao C do SVM variando de 0.5 ou 1.

Os melhores resultados foram obtidos com 'C': 0.5, 'penalty': 'l2', 'solver': 'newton-cg'

Lendo a documentação do Sci Kit learn sobre a função de Regressão Logística, presumimos que os parâmetros que teriam maior influência sobre a acurácia do modelo seriam o parâmetro C, pois era um parâmetro familiar do modelo SVM e que possuía grande peso no modelo, e o parâmetro solver que possibilitava escolher valores que se encaixavam com o perfil da base, otimizando os resultados.

Gaussian Naive Bayes

```
names = ["GaussianNB", "BernoulliNB", "MultinomialNB"]

classifiers = [GaussianNB(), BernoulliNB(), MultinomialNB()]

count = 0

for nome, clf in zip(names, classifiers):
  model = clf
  #treinando o modelo
  model.fit(X_train, y_train)

  #predição
  y_pred = model.predict(X_test)

  #Resultados do classificador
  print(nome)
  print(classification_report(y_test, y_pred))
```

Nesse caso não foi aplicado o grid-search, mas sim um comparativo entre o desempenho de três tipos de Naive Bayes: o Gaussiano, de Bernoulli e Multinomial.

O modelo Gaussiano obteve desempenho melhor entre os três, mas ficou muito distante em relação aos outros algorítmos analisados.

MLP

```
model = MLPClassifier()

parameters = {'learning_rate': ['invscaling', 'constant', 'adaptive'],
              'activation':['identity', 'logistic', 'tanh', 'relu'],
              'hidden_layer_sizes':[(16,26)]
              }
```

Foram avaliadas as taxas de aprendizado invscaling, constante e adaptativa, as funcões de ativação dos tipos indentity, logística, tanh e relu, e a tupla (16,26) para cadas ocultas representando 16 atributos e 26 classes.

Os melhores resultados foram obtidos com a função de ativação tanh e a taxa de aprendizado do tipo invscaling.

No contexto do perceptron, um dos parâmetros mais importantes para a qualidade da classificação é a função de ativação, por tanto, selecionamos esse parâmetro para a avaliação no MLP. Outro parâmetro importante é a escala da taxa de aprendizagem que vai influenciar na velocidade de convergência e precisão do modelo.

SVM

```
model = SVC()

parameters = {'C':[1, 20],
              'kernel':['linear','rbf','sigmoid'],
              'decision_function_shape':['ovo']
              }
```

Foram avaliados os parâmetros C variando de 1 ou 20, os kernels linear, rbf e sigmoid e a função de decisão do tipo “one versus one”

Os melhores resultados foram obtidos com 'C': 20, 'decision_function_shape': 'ovo', 'kernel': 'rbf'

O parâmetro C regula a penalidade, afetando a precisão no classificador. Outro parâmetro avaliado é o kernel que define a forma de divisão do classificador. Optamos pela função de

decisão do tipo “one versus one” para evitar gerar desbalanceamento devido a quantidade de classes diferentes existentes na base.

4. Conclusões

Nesse relatório avaliamos 7 modelos diferentes de algoritmo de aprendizado supervisionado e coletamos métricas para poder avaliar seus respectivos desempenhos.

Os 2 piores desempenhos ficaram com os modelos gerados com a Regressão Logística e Naive Bayes Gaussiano.

O fator relacionado ao baixo desempenho na classificação do NB pode estar relacionado à quantidade features avaliadas e o balanceamento da base. Era esperado que o modelo Multinomial desempenhasse melhor do que o Gaussiano pela quantidade de classes, mas não ocorreu. Também é preciso considerar que não foram testados ajustes de parâmetros para o comparativo de modelos de Naive Bayes.

Com relação à Regressão Logística, o parâmetro C recebeu valores iguais ou superiores ao C do SVM, gerando menor penalidade no classificador e por consequência menor precisão. Algo que foi notado apenas após a conclusão dos testes.

Por outro lado, aqueles que melhor desempenharam na classificação, foram os modelos KNN e SVM.

O KNN é um modelo mais simples e com parâmetros com efeitos mais claros sobre as classificações. Ele é indicado para bases balanceadas, que é o caso da letter, e possui muitos vizinhos próximos para cada ponto de dados. Atribuímos a esses fatores supracitados o seu bom desempenho.

O SVM foi um dos algoritmos mais demorados para execução, provavelmente devido ao maior rigor no parâmetro C de penalidade e a base ser multiclasse, exigindo várias rodadas de execução para comparar classes. Apesar da maior demora de execução, foi o modelo mais eficiente na classificação.

A escolha dos parâmetros que mais contribuem para o melhor/pior desempenho do modelo, pode ser uma oportunidade de melhoria nos nossos modelos construídos.

5. Referências

https://archive.ics.uci.edu/dataset/59/letter+recognition

https://academiatech.blog.br/matriz-de-confusao/#:~:text=O%20que%20%C3%A9%20a%20Matriz,os%20valores%20reais%20dos%20dados.

https://www.alura.com.br/artigos/metricas-de-avaliacao-para-series-temporais#:~:text=O%20erro%20quadr%C3%A1tico%20m%C3%A9dio%2C%20MSE,desses%20erros%20quadr%C3%A1ticos%20%C3%A9%20calculada.

https://www.elastic.co/pt/what-is/knn

https://scikit-learn.org/

https://ichi.pro/pt/hiperparametros-svm-explicados-com-visualizacoes-22257741819931

https://www.ibm.com/topics/