SEMINARIO DE SOLUCION DE PROBLEMAS DE INTELIGENCIA ARTIFICIAL 2

Practica 2. Ejercicio 2 y 3
Diego Campos Peña
15/05/2024

Portugal Pacheco, Juan Carlos
219747433
ING. COMPUTACION

Introducción:

Realizar una investigación sobre los siguientes métodos de clasificación en aprendizaje máquina:

- Regresión logística (Logistic Regression)
    - Este es un método de aprendizaje supervisado que se utiliza comúnmente para problemas de clasificación binaria, es decir, para predecir si una observación pertenece a una de dos categorías posibles. A pesar de su nombre, la regresión logística se utiliza para problemas de clasificación y no de regresión. La regresión logística utiliza una función logística, a menudo llamada "curva sigmoide", para modelar la probabilidad de que una observación pertenezca a la clase positiva en función de sus características.
- K-Vecinos Cercanos (K-Nearest Neighbors)
    - Este es un método de clasificación y regresión en aprendizaje automático. Se utiliza principalmente en problemas de clasificación, aunque también puede aplicarse a problemas de regresión. K-NN es un enfoque basado en instancias, lo que significa que toma decisiones en función de la proximidad a otras observaciones en el espacio de características.
- Maquinas Vector Soporte (Support Vector Machines)
    - Estos son un conjunto de algoritmos de aprendizaje automático utilizados tanto en tareas de clasificación como de regresión. SVM se destaca por su capacidad para trabajar eficazmente en espacios de alta dimensionalidad y su capacidad para encontrar límites de decisión óptimos que maximizan el margen entre clases.
- NaiveBayes
    - Este es un algoritmo de aprendizaje supervisado ampliamente utilizado en el campo del aprendizaje automático, especialmente en tareas de clasificación de texto y procesamiento del lenguaje natural. El enfoque se basa en el teorema de Bayes y asume independencia condicional entre las características, lo que lo hace "ingenuo" o "naive" en su suposición, ya que, en la realidad, las características pueden estar correlacionadas. A pesar de esta suposición simplificada, Naive Bayes a menudo funciona sorprendentemente bien en una variedad de aplicaciones.

Los métodos de clasificación buscan crear un modelo que pueda aprender y generalizar a partir de ejemplos de entrenamiento, de modo que pueda predecir la categoría a la que pertenecen nuevos datos.

El aprendizaje automático y la clasificación desempeñan un papel crucial en la automatización de tareas que involucran la toma de decisiones basada en datos, lo que les permite a las máquinas realizar tareas de clasificación con precisión y eficiencia.

Swedish Auto Insurance Dataset:

K-Vecinos Cercanos

```
Vecinos_Cercanos_Wine_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - Ejercicio 2 y 3/Wine Quality
Dataset')
Accuracy: 0.78
Matriz de Confusión:
[[677  75]
 [145  82]]
Precision: 0.52
Sensitivity: 0.36
Specificity: 0.90
F1 Score: 0.43
```

Regresión logística:

```
Maquina_Vector_Pima_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - Ejercicio 2 y 3/Pima Indians Diabetes Dataset')
Accuracy: 0.64
Precision: 0.00
Recall: 0.00
F1 Score: 0.00
```

Maquinas Vector Soporte

```
Maquina_Vector_Pima_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - Ejercicio 2 y 3/Pima Indians Diabetes Dataset')
Accuracy: 0.64
Precision: 0.00
Recall: 0.00
F1 Score: 0.00
```

NaiveBayes

```
Naive_Bayes_Wine_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - Ejercicio 2 y 3/Wine Quality Dataset')
Accuracy: 0.71
Matriz de Confusión:
[[538 214]
 [ 68 159]]
Precision: 0.43
Sensitivity: 0.70
Specificity: 0.72
F1 Score: 0.53
```

Pima Indians Diabetes Dataset:

● K-Vecinos Cercanos:

```
Cercanos_Pima_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - Ejercicio 2 y 3/Pima Indians Diabetes Dataset')
Accuracy: 0.70
Precision: 0.59
Sensitivity (Recall): 0.55
Specificity: 0.79
F1 Score: 0.57
Confusion Matrix:
True Positives: 30, False Positives: 21
False Negatives: 25, True Negatives: 78
```

```python
def find_neighbors(X_train, x_test, k):
      neighbors = []
      for i, x_train in enumerate(X_train):
            distance = euclidean_distance(x_train, x_test)
            neighbors.append((i, distance))
      neighbors.sort(key=lambda x: x[1])
      return neighbors[:k]
```

● Regresión logística:

```
Regresion_Logistica_Pima_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - Ejercicio 2 y 3/Pima Indians Diabetes Dataset')
Accuracy: 0.56
Precision: 0.43
Recall: 0.75
Specificity: 0.45
F1 Score: 0.55
```

```python
def train_logistic_regression(X_train, y_train, learning_rate,
num_iterations):
    m, n = X_train.shape
    weights = np.zeros(n)
    bias = 0
```

```
    for i in range(num_iterations):
        z = np.dot(X_train, weights) + bias
        predictions = sigmoid(z)
        error = predictions- y_train
        gradient_weights = np.dot(X_train.T, error) / m
        gradient_bias = np.sum(error) / m
        weights-= learning_rate * gradient_weights
        bias-= learning_rate * gradient_bias
    return weights, bias
```

- Maquinas Vector Soporte

```
Maquina_Vector_Pima_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - Ejercicio 2 y 3/Pima Indians Diabetes
Dataset')
Accuracy: 0.64
Precision: 0.00
Recall: 0.00
F1 Score: 0.00
```

```
def train_svm(X_train, y_train, C, gamma, max_iter):
    m, n = X_train.shape
    alpha = np.zeros(m)
    b = 0
    kernel = rbf_kernel(X_train, X_train, gamma)

    for _ in range(max_iter):
        for i in range(m):
        Ei = np.dot(alpha * y_train, kernel[i]) + b- y_train[i]
        if (y_train[i] * Ei <-0.001 and alpha[i] < C) or (y_train[i] *
            Ei > 0.001 and alpha[i] > 0):
            j = np.random.choice(list(range(i)) + list(range(i+1, m)))
            Ej = np.dot(alpha * y_train, kernel[j]) + b- y_train[j]
            alpha_i_old, alpha_j_old = alpha[i], alpha[j]
            if y_train[i] != y_train[j]:
                L = max(0, alpha[j]- alpha[i])
                H = min(C, C + alpha[j]- alpha[i])
            else:
                L = max(0, alpha[i] + alpha[j]- C)
                H = min(C, alpha[i] + alpha[j])
            if L == H:
                continue
            eta = 2 * kernel[i, j]- kernel[i, i]- kernel[j, j]
            if eta >= 0:
                continue
            alpha[j] = alpha[j]- (y_train[j] * (Ei- Ej)) / eta
            alpha[j] = min(H, max(L, alpha[j]))
            if abs(alpha[j]- alpha_j_old) < 0.00001:
                continue
            alpha[i] = alpha[i] + y_train[i] * y_train[j] * (alpha_j_old- alpha[j])
            b1 = b- Ei- y_train[i] * (alpha[i]- alpha_i_old) *
            kernel[i, i]- y_train[j] * (alpha[j]- alpha_j_old) * kernel[i, j]
            b2 = b- Ej- y_train[i] * (alpha[i]- alpha_i_old) *
```

```
            kernel[i, j]- y_train[j] * (alpha[j]- alpha_j_old) * kernel[j, j]
            if 0 < alpha[i] < C:
                b = b1
            elif 0 < alpha[j] < C:
                b = b2
            else:
                b = (b1 + b2) / 2
return alpha, b
```

- NaiveBayes

```
Naive_Bayes_Pima_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART
Dataset')
Accuracy: 0.75
Precision: 0.67
Sensitivity (Recall): 0.56
Specificity: 0.85
F1 Score: 0.61
```

```
def predict_naive_bayes(X_test, prior_probs, class_means, class_stds):
    n_classes = len(prior_probs)
    n_samples = X_test.shape[0]
    predictions = np.zeros((n_samples, n_classes))
    for c in range(n_classes):
        class_prob = np.zeros(n_samples)
        for i in range(n_samples):
            likelihood = np.prod(gaussian_probability(X_test[i],
            class_means[c], class_stds[c]))
            class_prob[i] = likelihood * prior_probs[c]
            predictions[:, c] = class_prob
        return np.argmax(predictions, axis=1)
def calculate_metrics(y_true, y_pred):
    tp = np.sum((y_pred == 1) & (y_true == 1))
    fp = np.sum((y_pred == 1) & (y_true == 0))
    fn = np.sum((y_pred == 0) & (y_true == 1))
    tn = np.sum((y_pred == 0) & (y_true == 0))

    accuracy = (tp + tn) / (tp + fp + fn + tn)
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
    specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
    f1_score = 2 * (precision * sensitivity) / (precision + sensitivity) if
    (precision + sensitivity) > 0 else 0
     return accuracy, precision, sensitivity, specificity, f1_score
```

Wine Quality Dataset:

- Regresión logística

```
In [27]: runfile('D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - E
Regresion_Logistica_Wine_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENC
Dataset')
Accuracy: 0.41
Matriz de Confusión:
[[204 548]
 [ 27 200]]
Precision: 0.27
Sensitivity: 0.88
Specificity: 0.27
F1 Score: 0.41
```

```
class LogisticRegression:
def __init__(self, learning_rate=0.01, num_iterations=1000):
    self.learning_rate = learning_rate
    self.num_iterations = num_iterations
def sigmoid(self, z):
    return 1 / (1 + np.exp(-z))
def fit(self, X, y):
    self.theta = np.zeros(X.shape[1])
    for _ in range(self.num_iterations):
        z = np.dot(X, self.theta)
        h = self.sigmoid(z)
        gradient = np.dot(X.T, (h- y)) / y.size
        self.theta-= self.learning_rate * gradient
def predict(self, X):
    return np.round(self.sigmoid(np.dot(X, self.theta))).astype(int)
```

- K-Vecinos Cercanos:

```
Vecinos_Cercanos_Wine_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - Ejercicio 2 y 3/
Dataset')
Accuracy: 0.78
Matriz de Confusión:
[[677  75]
 [145  82]]
Precision: 0.52
Sensitivity: 0.36
Specificity: 0.90
F1 Score: 0.43
```

```
class KNN:
    def __init__(self, k=5):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
    def predict(self, X_test):
        predictions = [self._predict(x) for x in X_test]
        return np.array(predictions)
    def _predict(self, x):
        distances = [euclidean_distance(x, x_train) for x_train in
        self.X_train]
```

```
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = np.argmax(np.bincount(k_nearest_labels))
        return most_common
```

● Maquinas Vector Soporte

```
Maquinas_Vector_Wine_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 -
Dataset')
Accuracy: 0.00
Matriz de Confusión:
[[  0   0]
 [226   1]]
Precision: 1.00
Sensitivity: 0.00
Specificity: 0.00
F1 Score: 0.01
```

```
class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01,
    n_iters=1000):
        self.learning_rate = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.weights = None
        self.bias = None
    def fit(self, X, y):
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0,-1, 1)
        self.weights = np.zeros(n_features)
        self.bias = 0
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.weights)
                self.bias) >= 1
            if condition:
                self.weights-= self.learning_rate * (2 *
                self.lambda_param * self.weights)
            else:
                self.weights-= self.learning_rate * (2 *
                self.lambda_param * self.weights- np.dot(x_i, y_[idx]))
                self.bias-= self.learning_rate * y_[idx]
    def predict(self, X):
        approx = np.dot(X, self.weights)- self.bias
        return np.sign(approx)
```

● NaiveBayes

```
Naive_Bayes_Wine_JEVH.py', wdir='D:/UNI/10MO SEMESTRE/SEM INTELIGENCIA ART 2/Practica 2 - Ejercicio 2 y 3/Wine Quality Dataset')
Accuracy: 0.71
Matriz de Confusión:
[[538 214]
 [ 68 159]]
Precision: 0.43
Sensitivity: 0.70
Specificity: 0.72
F1 Score: 0.53
```

```python
class NaiveBayes:
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.classes = np.unique(y)
        self.mean = np.zeros((len(self.classes), n_features))
        self.std = np.zeros((len(self.classes), n_features))
        self.priors = np.zeros(len(self.classes))
        for idx, c in enumerate(self.classes):
            X_c = X[y == c]
            self.mean[idx] = X_c.mean(axis=0)
            self.std[idx] = X_c.std(axis=0)
            self.priors[idx] = len(X_c) / n_samples
    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)
    def _predict(self, x):
        posteriors = []
        for idx, c in enumerate(self.classes):
            prior = np.log(self.priors[idx])
            class_conditional = np.sum(np.log(gaussian_probability(x,
            self.mean[idx], self.std[idx])))
            posterior = prior + class_conditional
            posteriors.append(posterior)
        return self.classes[np.argmax(posteriors)]
```

Conclusión:

la selección del algoritmo adecuado para un proyecto específico es fundamental para obtener resultados óptimos. La naturaleza de los datos y los objetivos del proyecto son factores clave a considerar al elegir entre diferentes algoritmos. Es importante tener en cuenta que los algoritmos evolucionan constantemente, por lo que es esencial mantenerse actualizado con las últimas investigaciones y ajustes. Comprender el código subyacente y realizar ajustes de hiperparámetros de manera sistemática son prácticas importantes para optimizar el rendimiento del algoritmo. En este sentido, la implementación del método de máquinas de vectores, con su robustez y capacidad para manejar problemas multiclase y relaciones no lineales, destaca como una opción prometedora para una amplia gama de conjuntos de datos. En última instancia, la interpretación de los resultados obtenidos es crucial para evaluar la eficacia del algoritmo seleccionado en relación con los objetivos del proyecto.

Bibliografía

● Webb, G. I., Keogh, E., & Miikkulainen, R. (2010). Naïve Bayes. Encyclopedia of machine learning, 15(1), 713-714.

● Mammone, A., Turchi, M., & Cristianini, N. (2009). Support vector machines. Wiley Interdisciplinary Reviews: Computational Statistics, 1(3), 283-289.

● Orea, S. V., Vargas, A. S., & Alonso, M. G. (2005). Minería de datos: predicción de la deserción escolar mediante el algoritmo de árboles de decisión y el algoritmo de los k vecinos más cercanos. Ene, 779(73), 33.

● Chitarroni, H. (2002). La regresión logística.

● Murphy, K. P. (2006). Naive bayes classifiers. University of British Columbia, 18(60), 1-8