Manual para el Examen de Keras - Clasificación Multiclase

Este manual te guiará a través de los pasos clave para construir y evaluar modelos de clasificación multiclase en Keras.

1. Preprocesamiento de Datos:

El preprocesamiento adecuado es fundamental para el éxito de tu modelo. Estos son los pasos a seguir:

- **Carga de datos:** Utiliza pandas.read_csv() para cargar tus datos desde un archivo CSV. Asegúrate de especificar el delimitador correcto si no es una coma. Usa df.head() para inspeccionar las primeras filas y df.info() para obtener información sobre los tipos de datos y los valores faltantes.
- **Manejo de valores faltantes:**
    - **Eliminar filas:** Si hay pocos valores faltantes, puedes eliminar las filas con datos incompletos usando df.dropna().
    - **Imputación:** Si hay más valores faltantes, considera imputarlos con la media, la mediana o la moda de la columna. Puedes usar sklearn.impute.SimpleImputer.
    - **Imputación con modelos:** Para datos mas complejos puedes usar modelos como KNNImputer, MissForest...
- **Codificación de variables categóricas:**
    - **Label Encoding:** Si el orden de las categorías es importante (variables ordinales), puedes usar LabelEncoder de Scikit-learn para convertirlas en números enteros.
    - **One-Hot Encoding:** Si el orden no es importante (variables nominales), usa OneHotEncoder para crear nuevas columnas binarias para cada categoría.
    - **Embeddings:** Si se trata de texto, tokenizar con Keras Tokenizer y pasar a la red una capa de Embeddings.
- **Normalización/Estandarización de variables numéricas:**
    - **Normalización (MinMaxScaler):** Escala los datos al rango [0, 1]. Útil cuando la distribución de las variables no es gaussiana.
    - **Estandarización (StandardScaler):** Resta la media y divide por la desviación estándar, resultando en una distribución con media 0 y desviación estándar 1. Generalmente, preferible para redes neuronales.

- **División de datos:** Divide tus datos en conjuntos de entrenamiento, validación y prueba usando train_test_split() de Scikit-learn. Usa stratify=y para mantener la distribución de clases en todos los conjuntos si es un problema de clasificación.

2. Construcción del Modelo:

- **Modelo Secuencial:** Keras proporciona la clase Sequential para construir modelos capa por capa.
- **Capas:**
    - **Capa de entrada (Input):** Define la forma de los datos de entrada (input_shape).
    - **Capas densas (Dense):** Capas totalmente conectadas. Especifica el número de neuronas y la función de activación.
    - **Capa de salida (Dense):** El número de neuronas en la capa de salida debe coincidir con el número de clases. Usa activation='softmax' para clasificación multiclase.
    - **Capas de Embedding (Embedding):** Si usas texto como entrada.
    - **Capa Flatten:** Aplana la salida del embedding para que sea un vector unidimensional.
- **Funciones de activación:**
    - **ReLU ('relu'):** Generalmente una buena opción para las capas ocultas.
    - **Softmax ('softmax'):** Para la capa de salida en clasificación multiclase. Produce probabilidades para cada clase.
    - **Sigmoid ('sigmoid'):** Para clasificación binaria en la capa de salida, o en otras capas para valores entre 0 y 1.
    - **Tanh ('tanh'):** Similar a Sigmoid, pero con valores entre -1 y 1.
    - **Lineal (sin activación o 'linear'):** Salida lineal, a menudo usada en la capa de salida para problemas de regresión.
- **Embedding:**
    - Para variables categóricas con alta cardinalidad o texto.

3. Compilación del Modelo:

- **Función de pérdida (loss):**

    - **'categorical_crossentropy':** Para clasificación multiclase con One-Hot Encoding en la variable objetivo.
    - **'sparse_categorical_crossentropy':** Similar a la anterior, pero para variables objetivo codificadas como enteros.
    - **'binary_crossentropy':** Para clasificación binaria.
- **Optimizador (optimizer):**
    - **'adam':** Generalmente un buen punto de partida.
    - **'sgd' (Stochastic Gradient Descent):** Puede ser útil con una tasa de aprendizaje bien ajustada.
    - **'rmsprop':** Otra alternativa.
    - Puedes pasar al constructor del optimizador el parámetro learning_rate.
- **Métricas (metrics):** ['accuracy'] es una métrica común para clasificación.

4. Entrenamiento del Modelo:

- **model.fit():** Ajusta el modelo a los datos.
    - X_train, y_train: Datos de entrenamiento.
    - epochs: Número de veces que el modelo ve todos los datos de entrenamiento.
    - batch_size: Número de muestras procesadas en cada paso del entrenamiento.
    - validation_data: Tupla (X_val, y_val) para evaluar el rendimiento en el conjunto de validación durante el entrenamiento.
    - validation_split: Si no se especifica validation_data, puedes usar validation_split para dividir automáticamente el conjunto de entrenamiento en entrenamiento y validación.
    - callbacks: Lista de callbacks para añadir funcionalidades durante el entrenamiento (e.g., EarlyStopping, ModelCheckpoint).
- **Callbacks:**
    - **EarlyStopping:** Detiene el entrenamiento cuando una métrica monitorizada deja de mejorar. Usa los parámetros monitor, patience y restore_best_weights.

    - **ModelCheckpoint:** Guarda los pesos del modelo durante el entrenamiento. Usa los parámetros filepath, monitor, save_best_only, save_weights_only, mode, verbose

5. Evaluación del Modelo:

- **model.evaluate():** Calcula la pérdida y las métricas en el conjunto de prueba.
- **model.predict():** Obtiene las predicciones del modelo. Usa np.argmax(predictions, axis=1) para obtener las clases predichas en clasificación multiclase.
- **Métricas:**
    - **Precisión (Accuracy):** Porcentaje de clasificaciones correctas.
    - **Matriz de confusión:** Muestra las clasificaciones correctas e incorrectas para cada clase. Usa sklearn.metrics.confusion_matrix y ConfusionMatrixDisplay.
    - **Informe de clasificación:** Proporciona precisión, recall, F1-score y soporte para cada clase. Usa sklearn.metrics.classification_report.
- **Visualizaciones:** Crea gráficos del historial de entrenamiento (pérdida y precisión) para visualizar el proceso de aprendizaje y detectar sobreajuste.

6. Ajuste de Hiperparámetros:

- **Manual:** Experimenta con diferentes valores para los hiperparámetros (número de capas, neuronas por capa, tasa de aprendizaje, dropout, etc.) y evalúa el impacto en el rendimiento del modelo.
- **RandomizedSearchCV / GridSearchCV (Scikit-learn):** Para automatizar la búsqueda de hiperparámetros. Debes usar KerasClassifier de scikeras para envolver tu modelo y especificar _estimator_type='classifier' en el constructor. Recuerda definir la función create_model para pasar los hiperparámetros como argumentos.

7. Técnicas para mejorar el Modelo:

- **Regularización:**
    - **L1/L2:** Añade una penalización a la función de pérdida para pesos grandes, lo que reduce el sobreajuste. Usa kernel_regularizer=l2(0.01) en las capas densas.
    - **Dropout:** Desconecta aleatoriamente neuronas durante el entrenamiento. Añade capas Dropout después de las capas densas.

- **Arquitectura del modelo:**
    - **Número de capas:** Experimenta con diferentes números de capas ocultas.
    - **Neuronas por capa:** Ajusta el número de neuronas en cada capa.
    - **Funciones de activación:** Prueba diferentes funciones de activación.
- **Optimizador:** Prueba diferentes optimizadores y ajusta la tasa de aprendizaje.
- **Aumento de datos:** Si tienes pocos datos, considera técnicas de aumento de datos para generar más muestras de entrenamiento.

8. Guardar y cargar el modelo:

- **model.save('nombre_del_archivo.keras'):** Guarda el modelo (arquitectura, pesos y configuración del optimizador).
- **keras.models.load_model('nombre_del_archivo.keras'):** Carga un modelo guardado.

Consejos Adicionales:

- **Comenta tu código:** Explica tus decisiones de diseño y los resultados obtenidos.
- **Analiza las métricas y las visualizaciones:** Interpreta los resultados y justifica tus conclusiones.
- **Experimenta:** Prueba diferentes enfoques y compara su rendimiento.
- **Simplicidad:** Empieza con un modelo simple y aumenta la complejidad gradualmente si es necesario.

Recuerda guardar tus modelos con el formato especificado por la profesora: APELLIDOSNOMBREinicial.keras, APELLIDOSNOMBREalternativa1.keras, etc.