ECE 612 - Applied Machine Learning Engineering - Summer 2025
Assignment 1: TensorFlow Data, Feature, and Keras API Basics

1 Aim of this Assignment

The goal of the assignment is to get you first-hand experience with

- Converting a "real" raw dataset into a tfrecord format that enables easy use within the TensorFlow Data API.
- Use keras preprocessing layers to handle basic feature pre-processing, including the embedding of categorical features and normalization of numerical features.
- Build custom preprocessing layers keras.
- Build, tune,and save a small keras model to infer a "real" quantity of interest from the dataset.

2 Raw Dataset and Your Objectives

For the purposes of getting your hands dirty with real-world heterogeneous data, you'll be working with a dataset that subsamples 8 years of historical hourly price information for a smallish collection of financial instruments including

- Exchanging the US dollar for the currency of a variety of other countries within a major foreign exchange trading platform.
- A number of exchange traded funds that track the major indices in the associated countries, and an ETF that attempts to track the change in the spot price of oil (light sweet crude delivered to Cushing, Oklahoma).

Additionally, a few major indices such as the S& P 500 index, the VIX volatility index, and an index tracking the yield on United States treasuries are included. For each of these, you have an open (the price that started the hour), a close (the price the hour ended at), a high (the highest price during the hour), and a low (the lowest price during the hour). The instruments involved have heterogeneous trading hours, so when a market is closed, the data from the most recent hour has been filled in. There are also missing values indicated by NaNs because of heterogenous dates among the securities when trading became available. Finally, there is a date attribute giving the date and time of the hour in question.

The simple prediction task we will look at is classifying the movement of the exchange rate between the US dollar and the Canadian dollar in the next hour based on the previous hours information for these securities. Precisely, the model will predict which of a series of discrete "bins" the fractional difference between the next hour's high and the present hour's close will lie in.

Please realize that there are ton of other relevant quantities that a economics course would indicate as being reasonably useful for predicting exchange rates such as this one that are not included among the features, for instance (1) sovereign debt yields in the countries, (2) some measure of corporate debt yield, (3) prices of related swaps and derivatives. Do not mistake this for a proper economics assignment - the point of this assignment is just to play with some real measured data from markets and practice operations within the tensorflow library, not to imbue an over-confidence of understanding how foreign exchange markets work.

More precisely, the dataset is meant to be an example of the type of heterogenous data you may encounter and want to work with within tensorflow and build a simple keras model for. We'll practice converting it into the tfrecord format, then reloading it, as well as handling the different types of features we'll create from it with appropriate keras preprocessing layers.

The dataset is shared with you as a pickle `appml-assignment1-dataset-v2.pkl` of a dictionary containing two pandas dataframes, which you can read with the pandas' read_pickle method. You'll read the pickle then extract the variables `X` and `y` from the python dictionary it returns. A given row of X gives the present value of the prices mentioned above, and the same row of y gives the next hour's Canadian dollar exchange rate high.

3 Tasks You Must Complete

In keeping with the purpose of the assignment described in Section 1, the tasks you must complete are separated into three main areas described in the subsections below.

3.1 Convert the Raw Data into a tfrecords File for Use with the Data API

Your first task will be perform some simple manipulations on the dataset then convert instances into Example objects to be saved with the example protobuf into a tfrecords file. All your manipulations and the following creation of the tfrecords file must be stored in a python file createSavedDataset.py. After loading the pickle containing the dictionary with the two Pandas dataframes, the code in createSavedData.py must perform the following manipulations on them

1. Use the numpy digitize command to quantize the fractional change of the next hour's CAD-high (in variable y) versus the previous hours CAD-close into 22 bins, based on 21 evenly spaced boundaries stretching from $-.001$ to .001 and including bins for being below $-.001$ and above .001. This will become the target/label for our classification task.

2. Create a 7 valued categorical attribute indicating the day of the week associated with the date attribute in the dataframe X.

3. Create a 24 valued categorical attributed indicatin the hour of the day associated with the date attribute in the dataframe X.

4. Create a 12 valued categorical attribute indicate the month of the year associated with the date attribute in the dataframe X.

Next the code in createSavedDataset.py must create an example protobuf for each instance must contain exclusively the following

1. A feature tickers which is the array of 188 numeric values of the original symbols this row of X (i.e. this instance) other than the date, sorted as a FloatList.

2. A feature weekday which is the day of the week for that instance, recognized as an Int64List.

3. A feature hour which is the hour of the day for that instance, recognized as an Int64List.

4. A feature month which is hte month of the year for that instance, recognized as an Int64List.

5. A feature target which is the index of the bin that the fractional change of the next-hour's high of this instance lies in, recognized as an Int64List.

Finally, the code in createSavedDataset.py must serialize each of these example objects into a string and written to a tfrecords file, named dataset.tfrecords containing example protobufs of the format described above for the entire dataset as manipulated above.

3.2 Utilize Keras Preprocessing Layers to Preprocess Numerical Features

Next, customImputeLayerDefinition.py must define a new class ImputerLayer creating a custom Keras preprocessing layer to replace NaN values with the min across instances without NaN in this position. An adapt method must calculate the minimum of the original non-NaN, values of each position in the vector input. Then the call method must replace the NaN values with the computed minimum.

The file buildAndTrainModel.py must then

1. Define a parse_example method which converts the serialized example protobufs back into dictionaries with the keys tickers, weekday, hour, month, target then returns an ordered pair of the dictionary without the target and the target for use with supervised training.

2. Load the dataset from the tfrecords file batch it with a batch size of your selection, and process it with the parse example method. Cache the result in memory using the cache command from the data API.

3. Split the loaded dataset into training, validation, and testing datasets.

4. Create a series of 4 keras inputs for the 4 labels in the instance dictionary (tickers,weekday,hour,month), specifying the appropriate datatype and shape.

5. Import the ImputerLayer, adapt it to the training data for the tickers attributes.

6. Create a Keras preprocessing normalization and adapt it to output of the imputer applied to the training data for the tickers attributes.

7. The tickers input should feed into the ImputerLayer followed by the Normalizer.

3.3 Build, Train, Evaluate, and Save a Keras Model

Next the file buildAndTrainModel.py must continue on to build and train the rest of the Keras model.

1. The categorical attributes weekday,hour,month inputs should be fed into Embedding layers of a dimension of your selection.
2. The outputs of the embedding layers and the normalizer should all be concat together to form an input to subsequent layers using tf.concat.
3. Include subsequent hidden layers as you wish, but the final output layer should have 22 outputs corresponding to the probabilities of the 22 bins of the fractional change and utilize softmax activation.
4. The keras model is then defined to stretch all the way from the inputs to the 22 outputs.
5. Compile and fit the model in a manner of your choice.
6. Save the model using the SavedModel format to mySavedModel.

You will then submit both your saved trained model and the code buildAndTrainModel.py utilized to create it.

4 What you must submit

Your submission should be a gzipped or zipped archive titled **abc123-lab1.tgz** or **abc123-lab1.zip** (with abc123 your drexel username). The archive must contain at least the following 4 files

1. createSavedDataset.py
2. customImputerLayerDefinition.py
3. buildAndTrainModel.py
4. mySavedModel.tgz or mySavedModel.zip a compressed archive of the saved model directory created by the model.save command.

5 How Your Assignment Will Be Graded

Your grade will consist of the following types of assessments which will be totaled to create a final score.

1. Compliance - was your submission named appropriately, and did it have all of the files.
2. Runs - does the code you submitted execute without errors.
3. Dataset - does your createSavedDataset.py code create the correct tfrecords file.
4. Imputer - does your customImputerLayerDefinition.py replace NaNs as explained above.
5. Loads Data - does your buildAndTrainModel.py load the data from your tfrecords and process it to produce a dataset properly.
6. Preprocessing - does your buildAndTrainModel.py create the Imputer and Normalizer preprocessing layers and adapt them.
7. Model Definition - does your model correctly recognize and process all inputs (including necessary embeddings and ultimate output layer) to produce the appropriate outputs.
8. Model Trained - does your code include compiling and fitting the model.
9. Model Saved - does your code include saving the model.
10. Model Match - does the saved model you submitted have a performance that well matches the model produced by re-running your code.
11. Model Performance - how does the accuracy obtained by your model on my held out testing data compare with that of your peers?
12. Compare your model's performance with a friend's model and explain why one model is performing better than the other.