APPLIED DATA SCIENCE PORTFOLIO

Kishan Polekar

SUID: 594650831 Email: kpolekar@syr.edu

Project 1: Baseball Pitch Prediction

The baseball pitch prediction project was a group project which utilized the built-in python 'pybaseball' dataset. This research aimed at exploring the Baseball (MLB) data and predicting what kind of a pitch will the pitcher throw based on several factors (explanatory variables).

Data

The dataset used for this report comes from the "pybaseball" package in Python. It consists of every single pitch thrown in the MLB since tracking has allowed. There were two parts to this dataset. The first one was the regular season data from the 2023 MLB season which we used as the training data for our model preparation. The next part was the test dataset. For this, we used the post season data (data from teams that qualified), and this helped us assess our models and select the best one for our predictions.

To collect the data, we simply installed the 'pybaseball' package available through Python (v3.9). Since this was a very large dataset, it took some time using a Google Colab CPU kernel, and we obtained around 717945 rows for the regular season alone. We then filtered the columns manually by handpicking a few columns that we thought would be helpful. We ended up with 32 columns (down from an initial 90+), and this was done through the 'drop' method in pandas. Finally, we identified 14 pitchers for both regular and post season data so that we can focus on a refined set of players. These players were selected based on number of games started during the postseason, and it helped us formulate better models while having enough datapoints. The final dataset had 38,456 rows in our training data and 4,532 rows in test data.

Preliminary Analysis

This section focused on visualizations that would help us get an idea of the data and figure out the model selection process or parameters we need to set for these models. It would also give us an idea about the relationship between various variables and factors that affect the outcome (pitch type). We used matplotlib and seaborn to get plots, graphs, and charts. We also made use of crosstab (pandas) that helped us group and sum the pitches by each player & pitch type.

For this section, we plotted some bar graphs which included both the X- and Y- axis labels along with a title. Then, we used a stacked bar graph with a colormap and denoted counts & proportions of each pitch type for each selected player. Next, we made use of seaborn for a similar task of plotting bar graphs. Since seaborn provides more flexibility with visualizations, we were able to group the bars for each player for the plot. We also performed Principal Component Analysis (PCA) and displayed the results using FacetGrid from the seaborn library. For this, we had to use matrix multiplication (numpy matmul),

eigen vectors (scipy eigh), transposing matrices, and combining all of it in a pandas Dataframe. PCA is available through scikit learn (sklearn.decomposition) and plotted as a scatterplot on a 2-D as well as 3-D map.

To apply the finishing touches to this section, we aimed at exploring the important variables that would help us predict our outcome variable by plotting a heatmap on our data. We made use of our regular season data, label encoded all the object type columns (string datatypes) using the LabelEncoder() function from sklearn (scikit-learn) preprocessing package. We were able to get a decent idea of the correlation between variables and focus on which columns to include in our models.

These visuals helped our audience which ranged from novice (who did not have any idea of baseball & very little idea of advanced python) to expert (who were able to figure out important details in a glance). All kinds of peers followed us through this initial analysis without any problems and we were able to capture our project's essence which was to present all our findings to every group without issues; be it knowledgeable or amateur.

Additional Insights

Before proceeding to our models and training, we explored our regular season data a little more through some more python machine learning libraries. Here, we performed item association using apriori (python package) as well as feature item sets. We used numpy arrays, pandas dataframes, lists, dictionaries, tuples, encoder fit transform, & TransactionEncoder function. We were able to get some association rules from left and right sides with confidence, support in the middle. Then, we explored the "Elbow Curve" to extract the number of important clusters needed to classify our datapoints in different pitch types. K-means method was used for different values of 'num_clusters' on our training dataset. Non-technical owners like to know which features associate with other features within the dataset, and through feature selection we can provide a direct answer that associates different variables and displays relations in our data.

Models: Training and Selection

This is where the magic happens. The main part of our program and analysis that uses machine learning & models available in python. A very powerful language provides us with multiple options for model selection including (but not limited to) K-means, Naïve Bayes, Random Forest, etc. We traversed through all of these to find the best parameters using the classic Grid Search algorithm provided by Python.

We tried out four different models and trained them on our regular season data using various parameters for each model. Then, we selected the best model out of our analysis, and used that for our predictions. Getting the models imported was easy since python already has all the ML libraries pre-installed, and we followed the exact same process with each one of them so that it is easy to compare & select. First, we initialized the models, then we set up the parameters for our grid search (each model having different settings),

then we fit the models on our training dataset (regular season). Once we had this, the next step was to select the best sub-model out of all these by getting the best parameters from all the options supplied, then displaying the best score (cross validation) so that it is comparable across the models. For evaluating our models, we used scikit-learn's metrics library which includes multiple functions like accuracy, f-1 score, precision, recall, etc.

We also explored another library (not pre-installed by python) called Optuna. This is like a grid search, but it is a lot faster and works with almost all models. The parameters that we set for optuna search are straightforward (easy to set up) with an optimizer that works on an objective to get the best result in quick time. We used this just for our last model (Support Vector Machine) while making use of the same metrics as previous models for evaluation. Once we had all the metrics, we displayed them in a tabular form which was easy to read and interpret by anyone.

Answering the question

The main goal of this project was to predict a pitch type. We tried to explore different models and fine tune different parameters within these models, and were able to achieve significantly good results, which were acceptable. We selected the best model out of all of these, which was the SVM, and use that model to predict pitch types on our test data. We predicted different pitches for all pitchers in the postseason by giving various input parameters, utilizing supervised learning as we knew what our target was in the test dataset. The best model gave us an accuracy of 35% in predicting the next pitch of a pitcher in the postseason and that is considerably good when accounting for numerous factors such as pitcher's health on that day, time of the innings, instantaneous weather, player's mood, batter's expression, their confidence, and so many more.

Project outcome

The course (Applied Machine Learning) aimed at providing us with the tools and knowledge to leverage different machine learning techniques, analyze datasets, gather required information regarding the data, cleaning it, providing it to some models, fine tuning the models, maintaining our train/test split so that our model can perform well on unseen data, overcome any obstructions during the model training phase, test out the 'goodness' of our models, and finally making predictions to provide actionable insights to business.

All through the course of the final project and the learnings within the class, we were able to accomplish all the above-mentioned objectives while performing efficiently as a team. There were several challenges that were encountered beginning from data collection up until our final prediction. Identifying those obstacles, analyzing the best approach to solve them, and being able to come up with quick & efficient solutions was an integral part of our learning. I can confidently say that I was able to make the most out of this course while applying my skillset to learn various new things that would be practically applicable in real world scenarios to help me solve business questions. The required skills for any job in today's market is well covered through individual & team efforts applied during the course.

Project 2: Text Classification

In this project, I fine-tuned a BERT (Bidirectional Encoder Representations from Transformers) model for text classification. The aim was to classify sentences or phrases into distinct categories, particularly focusing on detecting similarities between two sentences. By leveraging the capabilities of BERT models in natural language processing, the project produced results to enhance automated response systems in industries such as public health and banking. The project utilized a pre-trained BERT language model and TensorFlow for developing a classification model that can understand contextual information and semantic relationships in text.

Data

The dataset chosen for training and evaluating the model is the GLUE (General Language Understanding Evaluation) MRPC (Microsoft Research Paraphrase Corpus), accessed through TensorFlow Datasets (TFDS). This dataset consists of pairs of sentences collected from diverse online news sources. Each sentence pair is annotated to indicate whether the sentences are semantically equivalent or not. This annotated information serves as the base for training the model, with the two primary class labels being 'equivalent' and 'not_equivalent.' The dataset was accessed through TensorFlow Datasets (TFDS), which is a widely used open-source machine learning library for acquiring and managing datasets. The maximum sequence length of training and evaluation (validation) data is 128 characters.

Preprocessing

The analysis required a couple of steps before the model could be trained. First, I gathered the data which is done through TFDS, and then I pre-processed the data so that it could be understood by the model. The pre-trained BERT model does not recognize raw text; hence two things were done before feeding it to the model obtained from the Model Garden:

1. The text was *tokenized* (split into word pieces) and converted to *indices*.
2. Then, the *indices* were packed into the format that the model expects.

In the first step, the sentence undergoes a process called tokenization. Tokenization involves breaking down the input text into smaller units, typically words or subwords. For BERT models, a specialized tokenizer is employed to handle this task. The BERT Tokenizer segments each word into smaller units known as tokens, and these tokens form the basic building blocks for the subsequent stages of processing. The raw text is encoded into numbers or ‘tokens’ for the model.
After tokenization, the individual tokens need to be organized in a structure that BERT can comprehend. BERT recognizes a specific format for input data, typically organized as a nest of tensors. This nesting includes essential elements like the input_word_ids, input_mask, and input_type_ids. The packed input is formatted according to the requirements of the

BERT model, ensuring that the information is presented in a way that allows the model to understand and process the input effectively.
Special characters and tokens, such as [CLS] (classification token), [SEP] (separator token), and [MASK] (mask token), play crucial roles in BERT's architecture. In this step, special characters within the text are tokenized. Additionally, a special tokens dictionary containing a substantial number of samples (30k+) is used. This dictionary includes pre-defined embeddings for special characters or entities, contributing to the model's ability to handle a broad range of language patterns and expressions.

By following these preprocessing steps, the raw text data was transformed into a format suitable for input into the BERT model, enabling effective training and classification of sentences.

Training the model

When building the model, first the config file for the pre-trained BERT model that the model will be based on needs to be downloaded. Then, the file is tested out on a sample of 10 pairs of sentences taken from the training data. This generates a output of the 'similarities' between the pairs, and this output is in the form of logits of the two label classes (targets). This shows that the pre-trained BERT model has successfully been downloaded and is ready for further use. The structure of the BERT model is as follows:

- Activation function (hidden layers): GELU
- num_classes: 2
- num_hidden_layers: 12
- hidden_dropout_prob: 0.1
- input_shape: (32,128)
- Size of the network: 768 nodes

The BERT model has an input layer that accepts data in the form of nest of tensors which was built in the previous steps. This is then passed through a couple of embedding layers, and then fed to a pooling layer that flattens our sentence pairs in the shape of (input_size1, input_size2, 768). After this, there is layer normalization and some dropout that is applied. Finally, after masking the input, it is passed through a series of 12 hidden transformer layers which in turn gives us an output through the dense layer. This output is in terms of logits which must be converted to actual probabilities later on.
When the model is set up, the encoding weights are randomly initialized. These weights are restored from a checkpoint using the tensorflow Checkpoint function and pass the bert_model's checkpoint parameter. Once this is done, the next step is the optimizer setup.

BERT usually uses the Adam optimizer with weight decay (AdamW) which is used for this analysis. It employs a concept of learning rate starting from 0, then 'warming up' to a certain value, and then gradually degrading/decaying back to 0. For the model, we will use a Polynomial Decay function, and a Linear Warmup scheduler. The optimizer also initializes the batch size for our training and evaluation dataset to 32 and the number of epochs to 5. For this model, I used accuracy as an evaluation metric and the

SparseCategoricalCrossEntropy loss function since I had a classification problem and almost all the sentences did not meet the maximum size requirement and need padding, so I had a lot of empty data when passing to the model. I tried out different values of batches and learning rates, but they did not seem to have a significant effect on the output when tested on the validation dataset (metrics compared: loss & accuracy).

The different learning rates tried out and compared against validation data for loss and accuracy through base model optimization resulted in about 68% accuracy.

After the hyperparameters were setup and fine-tuned, I compiled the model and fit it to the training data. I tested out the results on both training and validation data. Over 5 training epochs, I achieved very good accuracies, and the losses also went down by a lot. As compared to the base BERT model that I downloaded earlier, my new model seemed to perform much better. The training accuracy went up from about 60% in the beginning stages to about 96% when I was done fitting the model to my training data.

Results

The main question I addressed in this project was how to fine-tune a BERT model for text classification, specifically focusing on detecting similarities between pairs of sentences. The project aimed to classify sentences into predefined categories, leveraging the capabilities of BERT models in natural language processing. The goal was to optimize the model's performance through hyperparameter tuning and training to enhance automated response systems in industries such as public health and banking.

The project involved data gathering, preprocessing, hyperparameter tuning, model training, and result analysis. Through hyperparameter optimization, the model achieved improved accuracy and lower validation loss compared to the base BERT model. Future steps include domain-specific classification expansion and GPU utilization for training efficiency. I used TensorFlow and BERT models for natural language processing tasks, aiming to enhance automated response systems and improve text classification accuracy.

Project outcome

The course (Artificial Neural Networks) dealt with latest techniques in the world of Artificial Intelligence. There were a lot of models, concepts, examples, and exercises that aimed at introducing us with Neural Networks. Other things in the course included hidden layers, depth of a network, transformers, GANs, LLMs, optimizers, forward & backpropagation networks, and other things that are necessary to build complex AI models.
The results and conclusions of the project indicated that the fine-tuned BERT model, trained using the GLUE MRPC dataset, successfully predicted similarities between pairs of sentences. By leveraging TensorFlow and BERT models for text classification, the project achieved improved accuracy and lower validation loss compared to the base BERT model. Using these tools, I will be able to interpret large neural network models and make valuable contributions to any firm since these technologies are popular in the current world.

Project 3: Order Fulfillment

The FudgeInc Order Fulfillment project is about merging two business entities into a single FudgeInc and analyzing the order fulfillment process for the company. The main research question was to identify factors leading to lag times in order fulfillment & suggesting possible solutions to reduce the same.

Data

Our initial data comes from two different sources: FudgeMart which is like an online shopping service like Amazon & FudgeFlix which is akin to Netflix. Both these datasets were stored in the form of databases accessed through SQL Server Management Studio. Each database had multiple tables which were filtered and selected for the merger in a bigger data warehouse ('FudgeInc') using Microsoft Visual Studio.

There were three steps in getting our data ready for analysis. First was to use some data modeling techniques to clean the data from both sources. Next, create and run an SQL script that would generate schemas & tables from our two databases for the merger into our final bigger database. Finally, use Visual Studio to design pipelines, clean some more data, automate the merge, and store the results in a data warehouse.

Data Setup

When looking at the data, we focused on a few tables from both the databases, and utilized them for the merger. Before preparing them for our final database, some data modeling was needed. This was done through MS Excel using a macro sheet that would generate SQL code later as needed. In the final schema, there were 4 tables: Date, Products, Customers, & Shippers. To fill these tables, we needed to select data from both the sources & use Visual Studio for data population.

The first table in our merger company FudgeInc is the Date table. For this, we didn't need to process our base tables a lot since all the fields were compatible with each other. The second table was Product. FudgeFlix was easier since it had fields from only one table 'ff_titles'. But for FudgeMart, we had to join two tables 'fm_products' & 'fm_vendors' to get the supplier's name. Our third table was Customer which included customer details from both platforms. Here, FudgeMart had data stored in a single table, but FudgeFlix data was gathered from two tables ('ff_accounts' & 'ff_zipcodes') for customer location (city, state, zip). Our final table was the Shipper table which was just the shipper's name from FudgeMart. We were skeptical whether or not to include this in our analysis but decided to keep it since we wanted to know if certain shippers have a higher lag time than others.

Lastly, all the fields from both tables were converted to similar data types, foreign keys were selected for all tables, a final 'lookup' table was created that contained lookup keys, a calculated field ('order_to_ship_lag_in_days') and a source column ('fudgemart' or

‘fudgeflix’). Once these things were finalized, a macro was used to generate SQL scripts for schema & table generation. It also added foreign key constraints, connected different fields & changed data types of some fields. Once we executed the SQL script, we were ready for the SSIS (SQL Server Integration Services) data insertion steps using Visual Studio.

Data Integration

This was the most important part of the project since we had to gather data from both our sources, apply transformations, get the schemas right, and dump all the data in our data warehouse. We made use of the SSIS package in Visual Studio and created packages, pipelines, & more.

There were three main packages for our ETL process. First one was to load the date dimension, second to stage all our order fulfillment tables, and third to dump the data to the data warehouse. For the DimDate table, we had 36,525 records. Next was the package to Stage Products, Customers, Shippers, OrderFulfill. Upon merging, the Products table contained 7238 rows, Customers had 60 rows, Shippers had 4 records & OrderFulfill had 12,834 records. After all these tables were staged successfully, they were ready for data warehouse dump. Upon successful completion of the integration, the next step was creating visualizations through PowerBI & answering some business questions.

PowerBI

Through PowerBI, we explored out data further using ROLAP and MOLAP. We identified the key factors impacting order processing and fulfillment times, such as operational staffing levels, weekend impact, order prioritization, and supplier interactions. We explored the influence of various economic, logistical, and regulatory factors within different states on lag time for order fulfillment. We mapped out the lag time affected by product category. Different product categories may have distinct characteristics, handling requirements, availability, and processing complexities, all of which can influence the lag time.

There were other visuals, dashboards, & stories that helped us answer some other questions like the average lag times across different product categories, which shippers take longer to get the products out, geographical impacts of orders, and times of the year deliveries take longer.

Answering the questions

The main goal of the Data Warehouse project for FudgeInc's Order Fulfillment was to analyze various aspects of the order fulfillment process to identify factors contributing to delayed shipments and order fulfillment inconsistencies. The project aimed to deliver insights and suggest possible solutions for reducing lag time in order fulfillment.
Some of the questions we attempted to answer through our analysis included:

- What are the customer trends based on specific orders or titles?
- Which type or category of products ships faster to customers?
- How do different shippers fare in terms of delivery speed and service quality?

- Is order fulfillment quicker for certain customers?
- How soon are certain orders fulfilled, and how does the lag time compare across different factors?

By addressing these questions and analyzing the data related to customer behavior, product performance, shipping efficiency, and order processing, we aimed to gain insights into the order fulfillment process at FudgeInc. The goal was to optimize inventory management, streamline order processing, prioritize order fulfillment, and improve overall operational efficiency to enhance customer satisfaction and reduce lag time in order fulfillment.

Project outcome

The course (Data Warehouse) started off by dealing with the basics of SQL, introducing tables, data types, joins, querying, and merging tables & databases. Then we moved on to using SSIS for data cleaning, integration, and dumping. We were familiarized with packages, star schemas, ROLAP & MOLAP techniques, and finally using PowerBI for informative/interactive visuals & insights.

The entire course along with the group final project enabled us to learn a lot about how ETL processes work, how to set up data for analysis, & create useful dashboards that will enable businesses answer key questions. All these skills are necessary in jobs today. Some challenges within the course were deliberately added so that we understand how to tackle them and come up with solutions. Utilizing the learnings from this course would prove to be beneficial for the overall development of my knowledge in the field of data science & data analysis.

Project 4: Healthcare Cost Analysis

The healthcare cost analysis project was a group project where we act as a consulting company for HMOs (Health Management Organizations), which are associations of health insurance companies that provide medical services in return for a predetermined yearly fee.

Our aim was to advice HMOs on how to cut costs to lower the HMO's overall health care costs. Our goal was to identify the key factors that determine why some people need more medical attention than others, identify those who will spend a lot of money on healthcare in the upcoming year, and identify those who will be expensive to the insurance companies.

Data

The data set used for this project was acquired from a Health Management Organization. The data frame consisted of 14 columns and approximately 7582 observations. The data set was downloaded as a .csv file and imported into R Studio, where it was stored in a new data frame named HMO_data. It had 6 numerical columns with a few of them containing some null values.

Data Cleaning & Analysis

First, data cleaning was carried out to ensure the data quality and conducted variable analysis to understand the characteristics of each variable in the data set. We ran functions such as summary() to analyze the statistical characteristics of each variable. Using statistical methods, we identified that there are NA values in variables such as bmi and hypertension. This might be due to manual errors while entering patient's information in the database. We used na_interpolation() function from the imputeTS package to remove the N/A values from the variables mentioned above. We analyzed the cost variable and identified that the outliers lie within the top and bottom 0.5% of the data. Using quantile() function, we removed the possible outliers.

Visualizations

In this section, we utilized the ggplot2 library to generate various graphs and charts that will help us analyze our data. We deployed descriptive statistics and visualizations to explore the data further and gain insights into the distribution and relationships among variables. We plotted bar graphs for each of the potentially significant variable against the cost variable. We analyzed that the smoker attribute has the most significance on the cost variable. Moreover, we plotted a scatterplot for age and bmi variables against the cost variable and observed that there exists a slightly linear relationship between them.

To visualize the impact of cost variable geographically, we plotted maps which explained the distribution of cost variable in the states included in the dataset. We found that the

state of Maryland has a higher average cost as compared to other states. Moreover, after creating the expensive variable, we visualized the data geographically by plotting maps to demonstrate distribution of population of expensive and non-expensive patients in various states.

Models: Training & Selection

Before running models on the data, we divided the dataset into training and testing datasets. The proportion that we selected was 70:30 i.e., 70% training data and 30% testing data.

Firstly, we applied a linear regression model to the dataset. Using this model, we were able to predict the cost variable with almost 59% accuracy using predictors such as age, smoker, exercise, bmi and hypertension.

Secondly, we used SVM model using the 'svmRadial' method. This is the most popular and most used method as it is very similar to a Gaussian distribution. We achieved an accuracy of around 93% using the SVM model. To get this accuracy number, we used the confusion matrix function from the caret package.

Thirdly, we used the K-SVM model. This is a Kernel-SVM approach. The difference here as compared to the normal SVM approach is that we specify the number of Kernels (points) that we will use closest to the current point to determine if they are similar or not for classification. This method uses the efficiency of SVM along with the accuracy of KNN (nearest neighbor) method. The accuracy that we achieved using this method was better as compared to SVM model, this one being around 98.36%. We used the confusion matrix function to observe the model accuracy.

Answering the questions

Our project for identifying underlying factors that affect healthcare costs (insurance premiums) & predicting 'expensive' patients utilized several libraries in the R language. We were able to read data, clean it, visualize it, and run models to make predictions. Following were the key takeaways from our analysis:

- People that are physically active on a regular basis are observed to have lower medical costs than those that do not.
- Regular exercise is also beneficial for mental health, as it helps to reduce stress and anxiety, improves mood and cognitive function, and promotes better sleep quality, all of which can lead to better overall health outcomes and potentially lower medical costs.
- Our analysis has highlighted that smoking has a significant impact on medical expenses. To address this issue, it is important to create awareness about the dangers of smoking and encourage individuals to quit smoking.

Project outcome

The course (Introduction to Data Science) was designed around a very in-demand language R, and focused on important terms, techniques, skills, & analyses required to successfully extract valuable insights from raw data. Through the use of semi-structured data

exploration tools available in R, we were able to organize, summarize, & process the data for analysis using graphs & models. These charts were properly labeled and self-explanatory. Next, the course introduced us to various machine learning models that can help us scrutinize our data, train models, and predict dependent variables.

All these methodologies enabled us to gain deep knowledge of the theories involved in data collection, cleaning, sorting, processing, modeling, & prediction. Furthermore, we were able to leverage eye-catching visuals through easy libraries which are in use in the job market at this time. I believe I learned a lot of things through the course and a group project was crucial in implementation of those learnings.