Classification

(**our process**) ***what we are doing***

is the process of creating a “thing-labeling” model **(f)** by training an algorithm to be able to correctly label “things” using pre-labeled things **(X)** *and* **(y)**

- We feed the model input variables **(X)** (features) and a target variable(s) **(y)**

Classification Predictive Modeling

(**model == function**) ***how we are doing it***

is the task of approximating:

- **(f)** a mapping / decision function ***from***
    - **(X)** input variables (data) ***to***
    - **(y)** discrete output variables (labels / answers)

Classifier / Classification Algorithm

(**what we are training**) ***our student***

is the algorithm (*series of repeatable steps*) that maps / classifies input data **(X)** to a specific class of **(y)**

- When the classifier is trained accurately (*using labeled input variables and a discrete output variable(s)*), it can be used to classify unlabeled data.

Lazy Learners simply store the training data and wait for the testing data to appear.

- Less training time, but take more time in predicting
    - KNN

Eager Learners construct a classification model based on the given training data before receiving validation and testing data for classification.

- Take a long time to train and less time to predict
    - Decision Tree

Step 1 | Acquire Data

acquire.py file

SQL

- use imported **get_connection( )** function to connect to Database
- use imported **get_database_data( )** function to read database into a DataFrame csv file
- **df = pd.read_csv('filename.csv')**

Google Sheet

- replace /edit with /export and add format=csv to beg of query string
- **csv_export_url = sheet_url.replace('/edit#gid=', '/export?format=csv&gid=')**

https//docs.googlecom/spreadsheets/d/BLAHBLAHBLAH ***/edit#gid=****NUMBER* *https//docs.googlecom/spreadsheets/d/BLAHBLAHBLAH* **/export?format=csv&gid=***NUMBER*

- **df = pd.read_csv(csv_export_url)**

Pydataset Import

- **df = data('db_name')**

Step 2 | Prepare Data

prepare.py file

Data Cleaning

- Format column_names
- Drop duplicate rows
    - use ***.drop_duplicates( )***
- Drop columns with too many null values
    - use ***.info( )*** or
    - ***.value_counts( )***
- Update datatypes
    - use ***.astype(*** datatype ***)***
- Encode binary categorical variables to numeric
    - use df ['encoded_column'] = df***.column.map(*** {'valuie_1' : 1, value_2: 0 } ***)***
- Create dummies to encode non-binary categorical columns
    - use:
        - dummy_df = pd***.get_dummies(*** df [['col_1', \ 'col_2', \ …]], dummy_na = False, \ drop_first = True***)***
        - df = pd***.concat(*** [df, dummy_df], axis = 1***)***
- **Any ideas for new features?**
- Handling outliers
    - Drop row(s)
    - Correct them to intended value
    - Create bins (cut, qcut)
    - Snap to selected min/max
- Scale numeric (continuous) features, if needed

Tidy Data

- Data should be tabular (made up of rows and columns)
- There is only value per cell
- Each variable should have its own column
- Each observation should have its own row

Melt use when one variable is spread across multiple columns
Wide ----> **Long**

- **df.melt(** id_vars = [‘col_no_melt’], var_name = ‘new_col_name’, value_name = ‘val_name’ **)**

Pivot use when one column contains multiple variables
Long ----> **Wide**

- **df.pivot(** index = ‘index_column’, columns = ‘col_to_pivot’**)**

Data Splitting

Train (in sample)

- Explore
- Impute values
- Scale numeric data (max( ) - min( ))
- Fit our ML algorithms
- Test our models

Validate (represents future, unseen data)

- Check for overfitting
- Testing our top models on unseen data and choosing best model for test

Test (out of sample, represents future, unseen data)

- Represents how we expect model to perform out in production

Step 3 | Explore Data

baseline prediction == most prevalent target variable class

- i.e. (titanic) “not survived”
- Will use to test usefulness of features in next step ---> modeline

Where we learn the “stories” and domain context contained in our data.

- Feature engineering
- Feature elimination to reduce noise
- Domain based outlier handling
  ^^^ *leads to better models*^^^

Other than a histogram or .value_counts( ), our exploration should only be done on our train dataset.

1. Identify and document initial hypotheses. (Use questions in the form of natural language)
2. Use visualizations to explore initial hypotheses.
3. When visualizations are not immediately clear or want additional evidence, use appropriate statistical test.

Univariate Stats

Normally done during prep, prior to splitting the data.
- Descriptive stats (numeric variables)
- Histograms and value_counts (discrete variables)

Finding outliers
Scaling variables

Bivariate Stats

Plotting the interactions of each variable with the target
- Numeric ----> numeric
  - Scatterplot
  - Lineplot
- Numeric ----> categorical
  - Barplot
  - Catplot
  - Boxplot
  - https://seaborn.pydata.org/tutorial/categorical.html

Explore the interactions of independent variables using visualizations and/or hypothesis testing to explore interdependence

Multivariate State

Asking additional questions of the data, such as how subgroups compare to each other and to the overall population.
- pairplot
- relplot
- catplot
- https://seaborn.pydata.org/tutorial/axis_grids.html

Which Hypothesis Test?

Pearson's R

Numeric ----> numeric
Linear relationship
corr, p = stats.pearsonsr(train_df.column, train_df.column)
https://ds.codeup.com/stats/more-statistical-testing-examples/#pearson-r

Spearman's R

Numeric ----> numeric
Non-linear relationship
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html

T-Tests

Numeric ---> categorical
https://ds.codeup.com/stats/compare-means/
Comparing sample mean with population mean
t, p = stats.ttest_1samp(train_df_sample, μ)

Comparing two sample means
t, p = stats.ttest_ind(train_df_sample1, train_df_sample2, equal_var = True/False)

ANOVA

Numeric -----> categorical
Comparing means of more than two groups
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html

Mann-Whitney u-Test

Numeric ----> categorical
Comparing means
Data does not match the assumptions of a t-test (*i.e. targets not normally distributed*)
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html

chi^2 Test

Categorical ---> categorical
Comparing proportions

- H(o)
- H(a)
- alpha ---> 0.05

https://ds.codeup.com/stats/compare-group-membership/
observed = pd.crosstab(a, b)

- **a = target**
- **b = feature**

chi2, p, degf, expected = stats.chi2_contingency(observed)

Document, document, document

- Initial question and assumptions
- Takeaways after each visualization
- Hypothesis tests:
    - H(o) and H(a)
    - Test run
    - takeaways
- Actions plan
    - Answers to initial questions
    - Next steps based on what you have learned

Step 4 | Modeling and Evaluation

df target and features DataFrame

- new features,
- additional cleaning

X_train features DataFrame

- feature selection,
- fit models,
- make predictions

X_train = train.drop(columns = ['target'])

y_train target series

- feature selection,
- evaluate model predictions

y_train = train.tagret

X_validate features DataFrame

- make predictions using top performing models

X_validate = validate.drop(columns = ['target'])

y_validate target series

- evaluate model predictions to access overfitting

y_validate = validate.tagret

X_test features DataFrame

- make predictions using best model

X_test = test.drop(columns = ['target'])

y_test target series

- evaluate model predictions made from X_test to estimate future performance on new data

y_test = test.target

0. Compute Baseline Prediction

```
train.baseline = y_train.mode( )
matches_baseline_prediction = y_train == baseline_value
baseline_accuracy = matches_baseline_prediction.mean( )
```

4a | Decision Tree

CART - Classification and Regression Trees

- We use the training data to train the tree to find a decision boundary to use as a **decision rule** for future data.
- Number of questions == depth of decision tree (too much ---> causes overfitting)

i. Create model object

```
clf = DecisionTreeClassifier( max_depth = x, random_state = 123)
```

ii. Fit the model

```
clf = clf.fit( X_train, y_train )
```

iii. Vizualize the Decision Tree

```
Dot_data = export_graphviz( clf, feature_names = X_train.columns,
                            class_names = clf.classes_,
                            rounded = True,
                            filled = True,
                            Out_file = None )
graph = graphviz.Source( dot_data)
graph.render( ‘df_decision_tree’, view = True)
```

iv. Make class predictions

```
y_pred = clf.predict( X_train)
y_pred[ 0:5 ]
```

v. Estimate probability of each target class (using training data)

```
y_pred_proba = clf.predict_proba( X_train )
y_pred_proba[ 0:5 ]
```

vi-00. Find optimal max depth

```
for i in range( 2, 21 ):
        # Make the model
        tree = DecisionTreeClassifier( max_depth=i, random_state=123 )

        # Fit the model (on train and only train)
        tree = tree.fit( X_train, y_train )

        # Use the model
        # We'll evaluate the model's performance on train, first
        y_predictions = tree.predict( X_train )

        # Produce the classification report on the actual y values and this model's predicted y value
        report = classification_report( y_train, y_predictions, output_dict = True )
        print( f"Tree with max depth of {i}" )
        print( pd.DataFrame( report ) )
        print( )
```

vi-01. Validate and Find Optimate Max Depth

Creates a DataFrame with columns:

- max_depth
- train_accuracy
- validate_accuracy
- difference

```
metrics = []

for i in range(2, 25):
    # Make the model
    tree = DecisionTreeClassifier(max_depth=i, random_state=123)

    # Fit the model (on train and only train)
    tree = tree.fit(X_train, y_train)

    # Use the model
    # We'll evaluate the model's performance on train, first
    in_sample_accuracy = tree.score(X_train, y_train)

    out_of_sample_accuracy = tree.score(X_validate, y_validate)

    output = {
        "max_depth": i,
        "train_accuracy": in_sample_accuracy,
        "validate_accuracy": out_of_sample_accuracy
    }

    metrics.append(output)
```

```
df = pd.DataFrame(metrics)
df["difference"] = df.train_accuracy - df.validate_accuracy
df0
```

4b | [Random Forest](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

i. Create the model object

ii. Fit the model

iii. Vizualize the Model

iv. Make Class Predictions

v. Estimate the Probability of Each Class (using training data)

4c | KNN

i. Create the model object

ii. Fit the model

iii. Make Class Predictions

vi. Estimate the Probability of Each Target Class (using training data)

4d | Logistic Regression

i. Create the model object

ii. Fit the model

iii. Feature Importance

Evaluate the importance, or weight, of each feature using the coefficients.
Evaluate the intercept of the model.

iv. Make Class Predictions

v. Estimate the Probability of Each Target Class (using training data)

Step 5 | Evaluation

5_0. Evaluating a classification model's performance

Document

- Positive
  - TP
  - FP
- Negative
  - TN
  - FN
- Consequences
  - FP (Type I Error) *Over confident*
    - Optimizing to minimize FP ---> **Precision**
  - FN (Type II Error) *Under confident*
    - Optimizing to minimize FN ---> **Recall**

5a. Compute Model Accuracy
5b. Create Model Confusion Matrix
5c. Create Classification Report