Machine Learning Engineer Nano-degree

Capstone Project-Mushroom Classification

Lin Zhu

Jan 31st, 2017

I. Definition

Project Overview

Mushroom hunting, also known as mushrooming or mushroom picking, describes the activity of gathering mushrooms in the wild, typically for eating. In common sense, generally mushrooms with bright color, special cap shape or wired smell are suspicious as toxic class, but is it all of the truth? (Figure 1)

Figure 1. Toxic Mushrooms

The pictures show three toxic wild mushrooms that are always mistakenly classified as edible food. They all looks normal but are fatal to human being. Therefore, how to identify the edibility of a certain types of mushroom?

In this project, I utilized the mushroom dataset from UCI machine learning repository, to train and compare multiple machine learning models. I have a special interest in this because I enjoy outdoor adventure, and mushrooms are delicious natural food available in wild. Therefore, it is important to learn how to tell a mushroom is edible or not.

The goal of our endeavors is to determine whether or not a mushroom is edible. Given death is an extreme consequence for guessing incorrectly, which is hard to accept, the optimal model should have least error on the edibility. Meanwhile, it is interesting to explore what features matter most in identifying the edibility.

Problem Statement

Goal

This study is about a classification problem. To fulfill it, I am going to design and evaluate a machine-learning classification model that could effectively identify the gill mushrooms' edibility:

- Clean and preprocessing "Mushroom.csv" dataset
- Split the processed dataset into training and testing dataset
- Apply machine-learning models to predict the mushroom edibility in testing dataset.
    - Utilizing Multilayer Perceptron Neural Network
    - Train models with the training dataset and make predictions of testing dataset
- Refine the model if any unsatisfied performance
    - Dimension reduction: Principal Components Analysis (PCA)
    - Adjustment in number of train steps
- Compare those evaluation metrics across models, and select the final model
- Which features are most indicative of a poisonous mushroom?
    - Explore the correlation between multiple features
    - Identify the top two features

Metrics

The label column contains binary label "e" (edible) and "p" (poisonous). Incorrect prediction especially labeling "toxic" mushrooms as "edible" might bring in fatal problems. According to intimal observation, with an "e" over "p" ratio of 1.07the dataset is balanced between "edible" and "poisonous".

- Accuracy score is commonly used in binary classifier evaluation. It considers both true positive prediction and true negative predictions with equal weight. Accuracy describes the closeness of the prediction to the true value,

- This project is about to predict the edibility. Accuracy returns the rate of all correct prediction (edible prediction for edible mushroom, poisonous prediction for poisonous mushroom) out of all type of predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN\ (the\ whole\ dataset)}$$

  - TP: true positive, the mushroom is edible, and predicted as edible
  - TN: true negative, the mushroom is poisonous, but predicted as poisonous
  - FP: false positive, the mushroom is poisonous, but predicted as edible
  - FN: false negative, the mushroom is edible, but predicted as poisonous
- Running time is another consideration. Sometimes there are trade off between the accuracy and running times. The differences in running time would increase along the data size extending. It would also generate high computing cost.

II. Analysis

Data Exploration

Dataset Overview

The UCI Machine Learning repository has a mushroom dataset, consisting of descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981).

The “Mushroom.csv” dataset has 8124 sets of data points. One column “Class” including “e” as edible and “p” as “poisonous”, as label. The other 22 feature columns are shown below:

- cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
- cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
- cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y
- bruises: bruises=t,no=f
- odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s

- gill-attachment: attached=a,descending=d,free=f,notched=n
- gill-spacing: close=c,crowded=w,distant=d
- gill-size: broad=b,narrow=n
- gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
- stalk-shape: enlarging=e,tapering=t
- stalk-root: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
- stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- veil-type: partial=p,universal=u
- veil-color: brown=n,orange=o,white=w,yellow=y
- ring-number: none=n,one=o,two=t
- ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
- spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
- population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
- habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

A sample data:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p

The dataset itself is well balanced between two type labels (Figure 3), with an edible-to-poisonous ratio of 1.07.

Figure 2. Balance check between “edible” and “poisonous” classes

Dataset Abnormalities

In the dataset, there are no NULL values. After an in-depth exploration, the feature “veil-type” contains just one unique value “p”, contributing little to the classification.

The feature “stalk-root” contains 2480 missing values filled as “?”. Considering the whole dataset only has 8124 values, I decide to keep rows with this type of missing values.

Exploratory Visualization

Distribution

Regarding each type feature, a joint plot below shows the data distribution by classes regarding each feature (The whole plot could be found in code, section “Data Preprocessing”, Figure 4 showed partial features). The data points are generally equally distributed between two classes, a little covariance regarding specific attributes. Some features draw special attention for unbalanced distribution of certain attributes, indicating differences between edible and toxic mushrooms. For the “Oder”, edible mushrooms are mostly no special smell, while poisonous ones have foul smelling. For the “spore-print-color”, edible ones are significant in “black”, but the “poisonous” one are with high quantity of “buff” and “chocolate”.

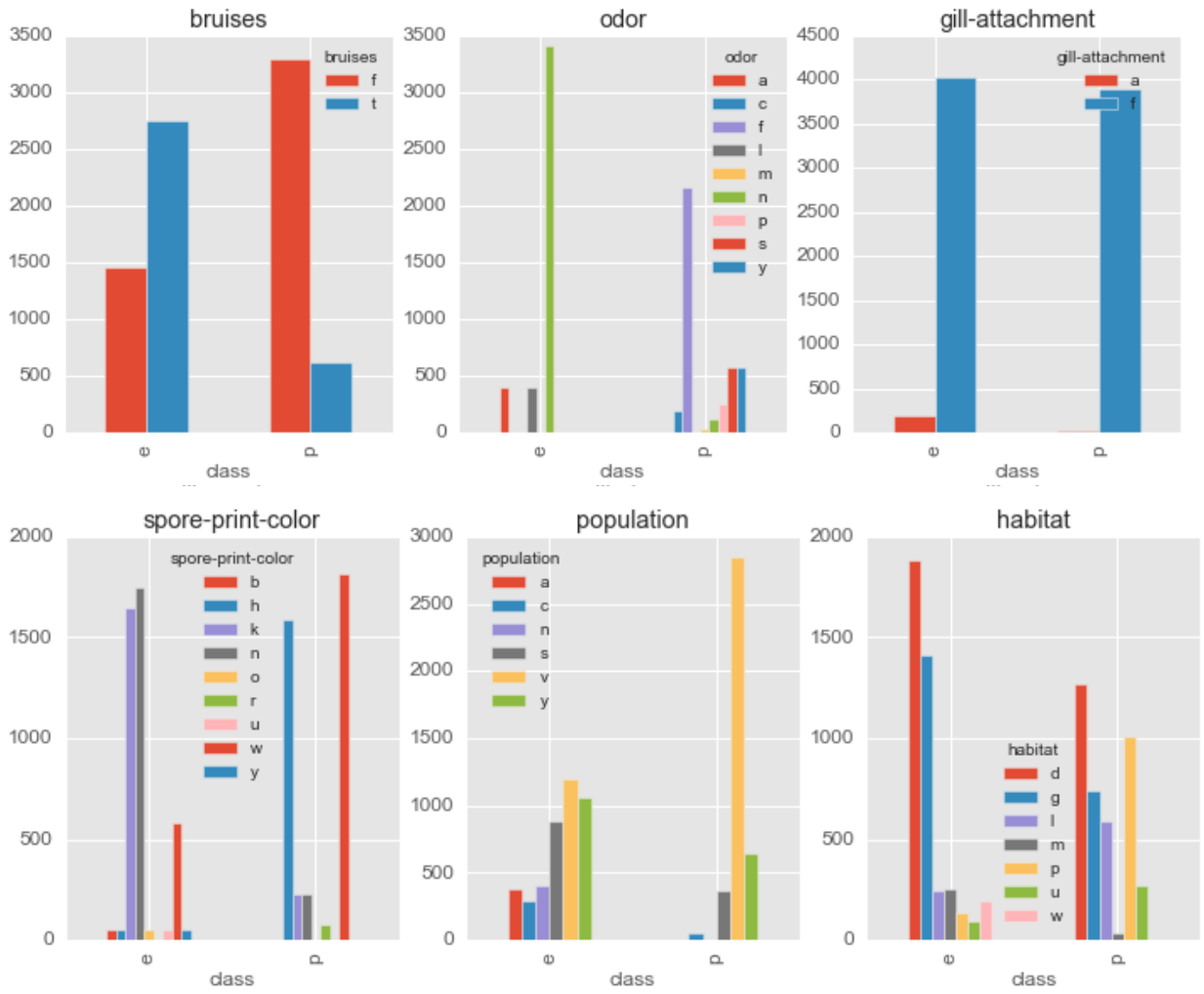

Figure 4. Distribution of selected features

Correlation

Those features could be further classified in subgroups like “cap”, “odor”, “gill”, “stalk”, “veil”, “ring”, “spore”, or by attributes, “color”, “shape”, “surface”, ect. As features in the same group might be correlated, a heatmap was generated regarding the covariance between multiple variables (21 features in total after “veil-shape” was removed).

According to the map, those deep red or deep green blocks indicate high covariance. There are around 15 deep color blocks showing the correlation between features. For example, “veil-color” vs “gill-attachment”, “odor” vs “ring-type”. Those specific correlations would be explored in future dimension reduction studies.

Figure 5. Covariance between 21 features

Algorithms and Techniques

The original dataset contains 21 valuable features that are all categorical data, which is not suitable to directly fit some machine learning models. The initial thought is to encode the features utilizing one hot encoder, and expand it into a 116 columns of dummy features. Considering the large scale of final feature sets (116 X 8124), I decided to try Multilayer Perceptron (MLP) neural network models with tensorflow.

The MLP-NN model architect (Figure 6):

- Number of layers:
    - Inputs layer (s)

    - Hidden layer (could be multiple)
    - Output Layer
- Implementing the regression
    - **Inputs and Placeholders**: use placeholder to define the shape of the inputs
    - **Weights**: the strength or amplitude of a connection between two nodes.
    - **Biases**: additional class of weights, it increases the capacity of the network to solve problems by allowing the hyper-planes that separate individual classes to be offset for superior positioning.
    - **Initializer**: create an operation to initialize variables just created.
    - **Model function**: how to compute our prediction.
- Train
    - **Loss function**: describe how inefficient our predictions are for describing the truth.
    - **Train step**: minimize the loss
    - **Number of steps**: how many times the model will be trained

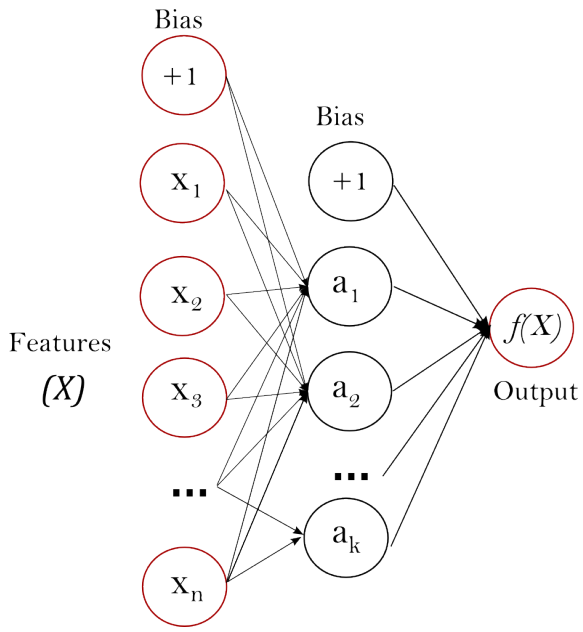

Figure 6. Typical MLP neural network

In the input layer, the set of neurons {xi|x1, x2, …xn} representing the input features, each neural in the hidden layer transform the values from the previous layer with a weighted linear summation, and followed by a non-linear activation function. The output layer receives the values from the last hidden layer and transforms them into output values.

Advantages of MLP:

- It is capable to learn non-linear models
- It is capable to learn real-time models

Disadvantages of MLP:

- MLP with hidden layer have a non-convex loss function with more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.
- MLP requires tuning a number of hyper parameters such as the number of hidden neurons, layers and iterations
- MLP is sensitive to feature scaling

Consider this project, MLP neural network model is suitable to handling the classification job with 21 or more features. To avoid its sensitivity in feature scaling, the One-Hot-Encoder was utilized in transforming categorical data into numerical data.

***Data preprocessing*:**

- **Data cleaning**: remove non-related data, missing value or treatments to any other abnormalities
- **Data encoding**: transform categorical data into numerical data for further training and prediction.
    - One-Hot-Encoder was adopted in this part. Compared to the Label Encoder, which is also quite frequently used, one-hot-encoder's advantages and disadvantages were shown below.
        - Advantages: it results in binary value rather tan ordinal and that everything sits in an orthogonal vector space.
        - Disadvantages: its cardinality. It blows up the feature space, and brings in the curse of dimensionality. One solution for it is to apply an PAC for dimension reduction

***Dimension Reduction*:**

- Principle Component Analysis: uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. Advantages and disadvantages for PCA:
- Advantages:
    - It could deal with large datasets

    - No special assumptions on the data and applicable to all datasets
- Disadvantages:
    - Non-linear structure is hard to model with PCA
    - The meaning of the original variables may be difficult to assess directly on latent variables.

Benchmark

This dataset was public available on Kaggle, the community already contributed multiple methods for edibility classification. Here an averaged accuracy value was selected as benchmark (refer to Reference 1-3, also solutions from link: https://www.kaggle.com/uciml/mushroom-classification/kernels?language=Python).
For example.

Model	Accuracy
SVC_linear	0.96
Logistic Regression	0.95
Random Forest	0.99
Decision Tree	1
SVC_rbf	1
Average Accuray	0.98

There is little discussion regarding running time, but I think computing cost is same important especial in industry, after chatting with people around regarding similar issues, I set a benchmark for the running time.

- For accuracy, my goal is to reach 0.98, optimally above 0.99
- For total running time, my goal is to reach 2s, optimally below 1s.

III. Methodology

Data Preprocessing

Data Cleaning

According to previous exploration, some abnormalities exist in the original dataset. Steps for data cleaning includes:

- Check the unique value of each features
- Remove features with only one unique value (remove "veil-type" column)
- Check any NULL, missing values ("stalk-root" has 2480 "?" value)

- Since “?” accounts for 30% of total data point, directly deletion of “?” values might bring in sharp reduction of data scale, I decide to keep it.
- After data cleaning, the original dataset was transformed into one with 21 features and 8124 rows.

Date Encoding

Based on previous analysis, the MLP neural network would be selected to train and predict mushroom edibility for testing side. The MLP neural network takes numerical inputs, thus an encoding method is in need to transform the categorical dataset into numerical ones.

I choose One-Hot-Encoder since it has the advantage that result is binary rather than ordinal and that everything sits in an orthogonal vector space. After One-Hot-Encoder, the clean dataset was transformed into a new one with 116 features.

One drawback for One-Hot-Encoder is the curse of dimensionality, if the result is not as expected, a dimension reduction scheme (e.g. PCA) will be utilized for further analysis.

Implementation

The implementation process includes two main stages:

- The classifier training stage
- The feature importance analysis stage

Task 1

For the training state, the classifier was trained on the preprocessed data. This was done on the Jupyter notebook, includes the following steps:

Load original dataset “mushroom.csv” into memory, conduct data exploration and applied preprocessing according to the “Data Preprocessing” section.

Generate training and testing datasets. Since the original dataset is with 8124 rows that are all labeled, a split ratio of 0.2 was set, indicating 20% of those data would be kept as testing set.

- Define model:
    - Input layer shape: [None, 116]
    - Output layer shape: [None, 2]
    - Define weight and biases
    - Prediction function: tf.sigmoid()
- Train and predict:
    - Loss function: cross_entropy()

- Train_step: gradient descent optimizer, with learning rate 0.5
- Feed function: use the training and testing dataset from step 2.
- Accuracy: tf.reduce_mean(), computer the mean over all the examples
- Number of steps: initial set as 300 iterations

Task 2

For the second task, the initial feature importance study, I decide to use PCA to explore and find an answer that, among all 116 expanded features (X_oh, the other two columns are label columns with dummy values, marked as y_oh), which ones really count. The study follows steps below:

- Import PCA
- Fit PCA with feature dataset (116 columns X 8124 rows)
- Get the explained variance ratio
- Plot the explained variance for each feature

The plot (Figure 7) below indicates that, the first 30 principal components retains 80% of the data, the first 55 principle components retains almost 90% of the data. Results also showed that the explained variance for the last 32 components is too small that could be treated as 0.

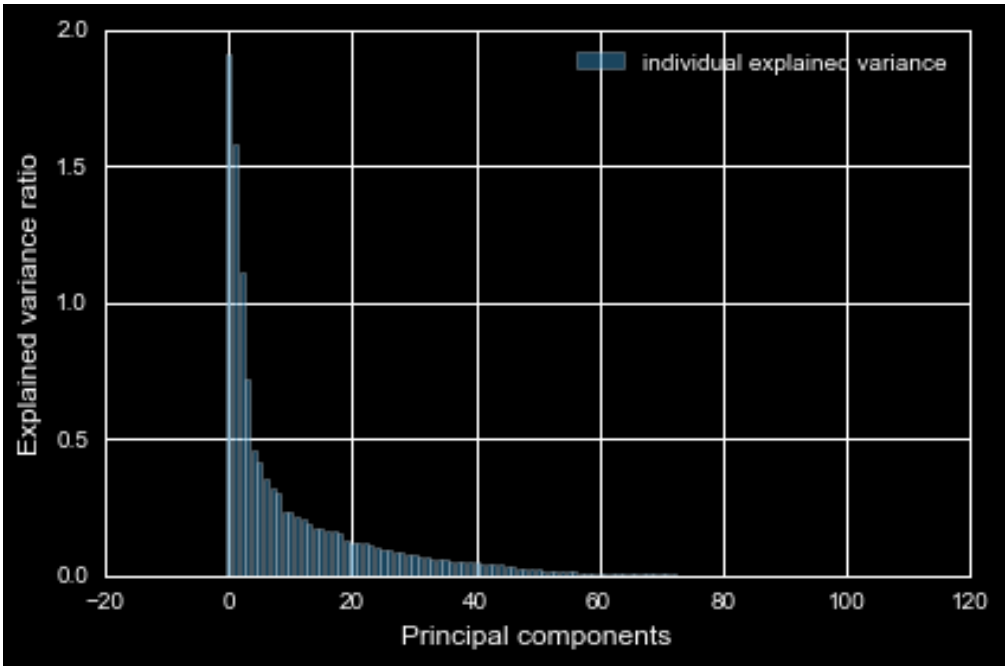

Figure 7. Principle components study

Complication Discussion

I never use tensorflow before. It took me some time to learn how to choose parameters and building MLP neural network models in tensorflow, especially for the loss function and prediction function. Later I decide to use the “sigmoid” as activation function. I learned the whole programming flow through the example case “MNIST For ML Beginners”. Overall, it is super fun time having my first try coding with tensorflow scheme, though lots of debugging through the whole process.

Refinement

Initial Solution Result

As benchmark listed, the average accuracy achieved by the community is 98%, with a running time within 2s. The initial run with 300 iterations got a set of result shown below:

- Accuracy: 97.48%
- Running time: 2.3s

Since both accuracy and running time exceed the benchmark threshold. More train steps were conducted for higher accuracy. This two plots below showed accuracy and running time under different number of iterations.

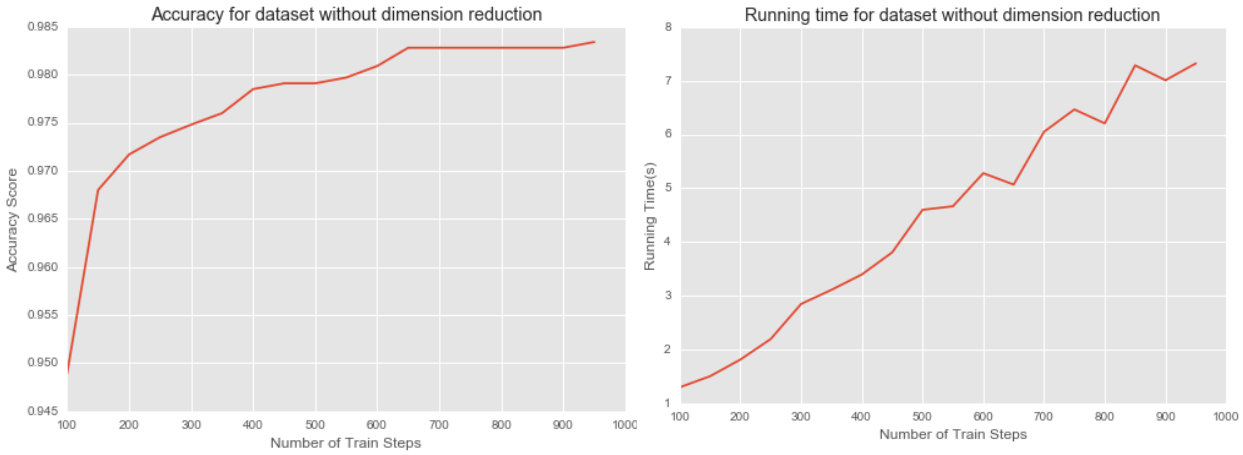

Figure 8. Accuracy and running time for initial model

The result (Figure 8.) showed that as training steps increase, the accuracy first increase and then stay almost the same when it is over 700 training steps, while the running time increase almost linearly with increase in training steps. With over 550 train steps, the model would achieve accuracy around 98%. However, it would take round 4.5 seconds, which exceed our requirements.

Improvement (PCA for dimension reduction)

As the feature importance plot showed, not every feature has equal importance. The first 30 components retain 80% dataset, and the first 55% retain 90% dataset. The model got improved through dimension reduction following steps:

- Set the number of components list for PCA fitting: n_components
- Fit feature columns (X_oh) with PCA (n_components), and transform feature dataset.
- Repeat the “Train” step for task 1 in the implementation section, except running more iteration combinations, e.g. range (100, 900, 100)
- Plot the accuracy and running time along the different n_components and running time combinations

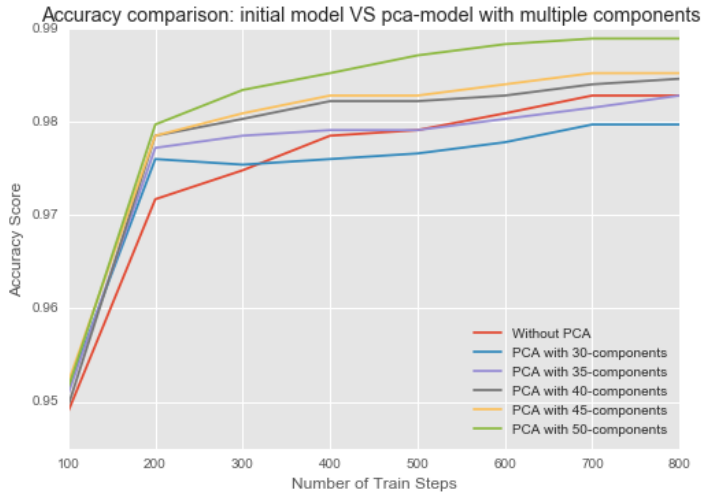

Figure 9. Accuracy comparison between initial model and feature reduction model

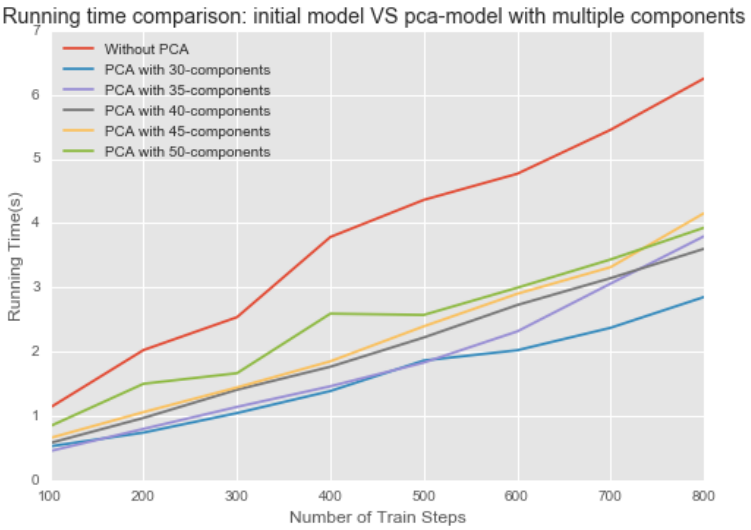

Figure 10. Running time comparison between initial model and feature reduction model

According to the comparison (Figure 9, Figure 10), with 50 principle components (green line), the new model almost achieved better prediction compared to the initial one (red line), around 1% improvement in accuracy under each train step. Meanwhile, the running time were cut by 30%-40%.

Task 2 Feature Importance Study

The previous feature importance study base on PCA is about how many features are important in the expanded dataset. To learn which feature from the original ones are interesting most, Xgboost was utilized to draw the feature importance chart for original dataset with 21 features.

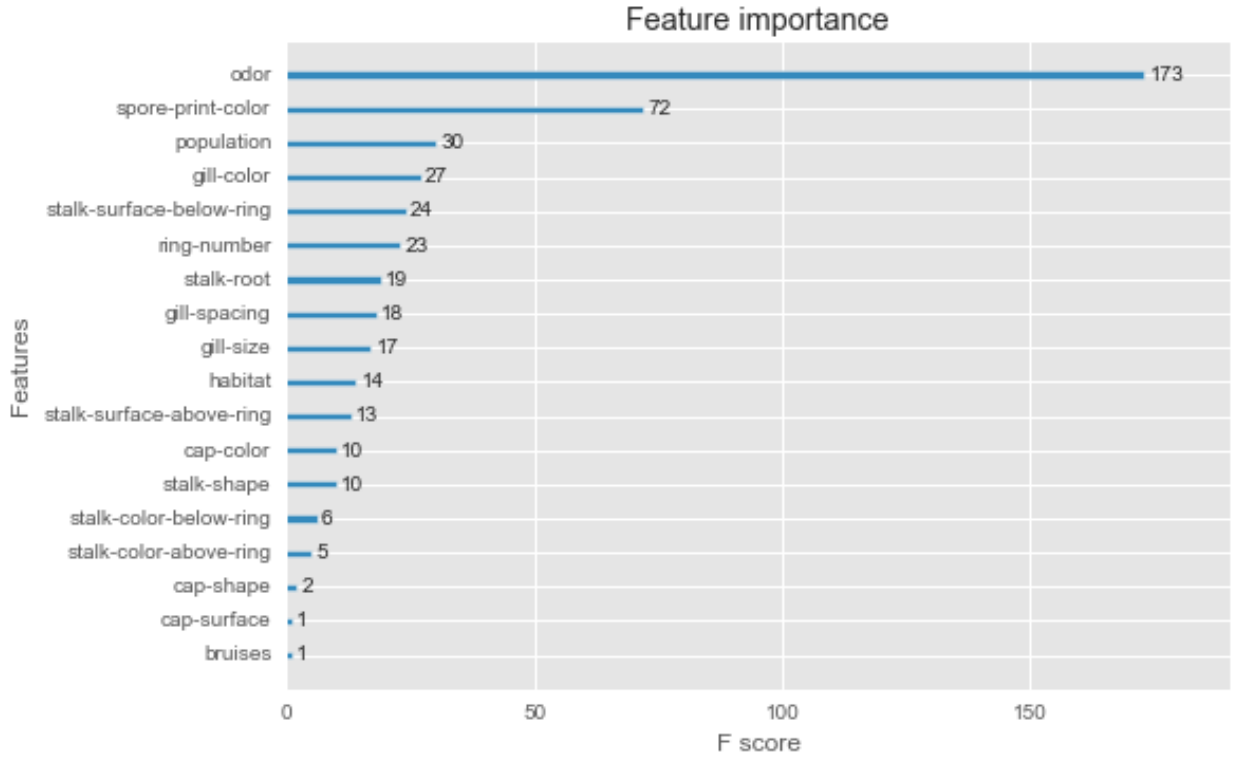

Figure 11. Feature importance study with Xgboost

The chart (Figure 11) above indicates that “odor” and “sprint” color are the top two most important features when determining if a mushroom is edible or poisonous.

IV. Results

Model Evaluation and Validation

Final Model

The final architecture and hyper parameters were chosen because they performed the best among all the combinations, regarding the accuracy and running time estimation.
A complete description of the final model are listed below:

- Number of layers: 2 layers
    - Inputs layer
    - Output Layer
- Implementing the regression
    - **Inputs and Placeholders**:
        - Inputs layer: [None, 50]
        - Outputs layer: [None, 2]
    - **Weights**: tf.zeros([50, 2]
    - **Biases**: tf.zeros[2]
    - **Initializer**: tf.global_variables_initializer()
    - **Model function**: tf.sigmoid

  Train
- **Loss function**: cross_entropy
- **Train step**: gradient descent optimizer, with learning rate 0.5
- **Number of steps**: 300

Robustness Test

In order to test the robustness, I sampled the expanded dataset with various sizes, and compare the variance in accuracy and running time among all those samples, as well as compare it with the full dataset as inputs. The test was conducted through following steps.

- Sample the expanded dataset (116 columns X 8124 rows) with various data size:
    - [2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000]
- Apply one hot encoder to each dataset
- Apply PCA to each dataset
- Create training and testing set with each dataset, using the same split ratio 0.2
- Train and prediction, with the same set of parameter as in the final model
- After statistical analysis, standard deviation was selected to test the robustness of the model with various inputs
    - Standard deviation for accuracy: 0.003
    - Standard deviation for running test: 0.18

The results shows accuracy changes little along changes in inputs. The variance in running time is larger because it is more sensitive to the size of training set. Overall, the model is quite robust and could be trusted.

Justification

Final Model Justification

The final result with 50 principle components, 300 iterations has the following result:

- Accuracy: 98.34%, it is a little bit better than the benchmark
- Running Time: 1.88s, it is also improved compare to the benchmark.

Generally, the accuracy and running time from final model is a little bit better compared to the benchmark. There are always trade of between accuracy and running time. With consideration of both, the final model listed above is the optimal solution. Given the accuracy higher than 98%, it is strong enough to solve the mushroom classification problems.

Feature Importance Justification

The second task is to identify top features, results showed that “Odor” and “Spore-print-color”, the following steps were used to justify the two identified features

- Reselected features only related to “Odor” and “Spore-print-color”, and save it as a new input feature set X_oh_feature
- Train and conduct with the same final model, keep everything the same except the number of features.
- The new model got results as below:
    - Accuracy: 97.67%, 0.6% lower compared to the result from final model
    - Running time: 1.76s, 6.3% faster compared to the result from final model

The above result is from only top two significant features related inputs. Considering there is only 0.6% accuracy variance, the top two features identified are sound and significant.

V. Conclusion

Free-Form Visualization

Below (Figure 12) is the accuracy comparison regarding initial model, final model with PCA, and model with only significant species related features.

After refinement, under 300 train steps, the final model (50 principle components) improves the accuracy by 1%, with a final result around 98.3%, allowing it strong enough to classify mushrooms.

For the revised final model with only “odor” and “print color” related features, its performance indicate that with train steps higher than 400, its accuracy will converge with the final model, indicating the importance of those two features are sound.

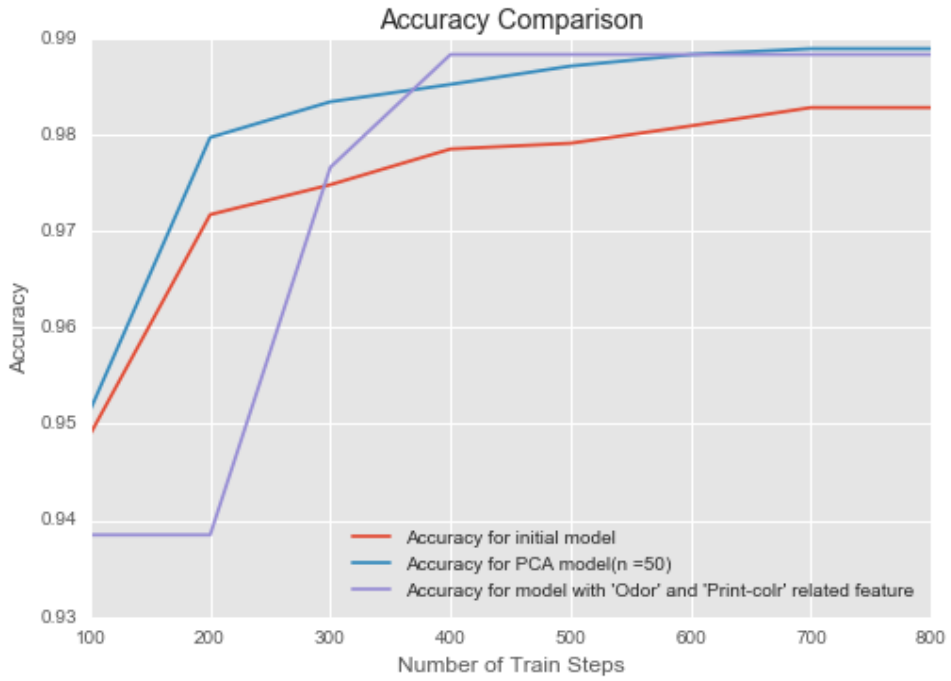

Figure 12. Accuracy comparison between initial model, final model and model with only significant feature related attributes

Though it got high accuracy, it should be noted that this model only suitable for classify the gill type mushrooms. The trends also showed that, under current model architect, it would significantly increase computing cost if we want to improve the accuracy by 1%. Overall, this model is applicable in limited circumstance with high accuracy.

Reflection

The process for this project can be summarized as following steps:

1. An initial problem was defined and a public available dataset were found and downloaded.
2. The data was preprocessed (Data cleaning, One Hot Encoder)
3. A benchmark was defined.
4. Choose a machine-learning model and build the classifier (tensorflow, MLP neural network).
5. Train the initial model with training set and prediction the testing set.
6. Refine the model utilizing PCA and find the best model that satisfies requirements on accuracy and running time, save it as final model.
7. Justified the model robustness with various sizes of inputs.

Step 4-6 is quite difficult to me. For step 4, I wasted tons of time tried different machine-learning models. Since the dataset is quite neat and clean, some default models like the Xgboost would directly give 100% accuracy. It also takes me time to learn the tensorflow API.

For step 5-6, I trained the model with a couple of (n-components, number of train steps) combination, though with tensorflow, it still took a couple of minute to run all of them. Once there is error, I need to wait for another few minutes to check the results.

Overall, it is very interesting to try neural network with tensorflow, I will do more in depth exploration later.

Improvement

In this project, I just tried neural network, and only use accuracy and running time as evaluation metrics. Some further steps could be tried later:

- Other machine-learning models:
    - Random forest, Xgboost, SVM
    - The “mushroom” dataset is comparatively clean and neat, 8124 row X 118 columns is not a huge dataset scale. Traditional machine-learning models might be more efficient in running this type of dataset, compared to neural networks.
- Use Label Encoder instead of One Hot Encoder
- Try different learning rate: here I only used one learning rate, 0.5. Exploration on how learning rate impact the model performance would also be interesting

- Add more metrics for evaluation: recall, ROC
- Try StratifiedShuffleSplit instead of train_test_split, enable cross_validation.

The model performance could also be improved by usage of deep learning library like keras. Overall, this project is pretty interesting and informative. Lots of techniques were applied and evaluated. I would implement those thoughts for further improvement

References:

1. Benchmark1:https://www.kaggle.com/acehanks/d/uciml/mushroom-classification/mushroom-poisonous
2. Benchmark2: https://www.kaggle.com/abangfarhan/d/uciml/mushroom-classification/predicting-mushrooms-and-dimensionality-reduction
3. Benchmark3: https://www.kaggle.com/nickmo464/d/uciml/mushroom-classification/quick-pass-at-mushroom-classification-ml
4. PCA: https://en.wikipedia.org/wiki/Principal_component_analysis
5. Github: https://github.com/linzhulz/student_intervention/blob/master/student_intervention.ipynb
6. Wiki: https://en.wikipedia.org/wiki/Mushroom_hunting
7. Toxic mushrooms: https://www.britannica.com/list/7-of-the-worlds-most-poisonous-mushrooms
8. Sequential model: @misc{chollet2015keras,title={Keras}, author={Chollet, Fran\c{c}ois}, year={2015}, publisher={GitHub},howpublished={\url{https://github.com/fchollet/keras}},}
9. https://www.tensorflow.org/tutorials/mnist/tf/
10. http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html
11. MLP neural network: http://scikit-learn.org/stable/modules/neural_networks_supervised.html
12.