UDACITY

Return to "Machine Learning Engineer Nanodegree" in the classroom

Finding Donors for CharityML

REVIEW

HISTORY

Meets Specifications

Very impressive submission here, as you have good understanding of these techniques and you now have a solid understanding of the machine learning pipeline. Hopefully you can learn a bit more from this review and wish you the best of luck in your future!

Exploring the Data

Student's implementation correctly calculates the following:

- **Number of records**
- **Number of individuals with income >$50,000**
- **Number of individuals with income <=$50,000**
- **Percentage of individuals with income > $50,000**

Would suggest looking into using pandas column slicing

```
n_greater_50k = len(data[data.income == '>50K'])
```

Another great idea would be to preform some extra exploratory data analysis for the features. Could check out the library Seaborn. For example

```
import seaborn as sns
sns.factorplot('income', 'capital-gain', hue='sex', data=data, kind='bar', co
l='race', row='relationship')
```

Preparing the Data

Student correctly implements one-hot encoding for the feature and income data.

Could also check out the apply method

```
income = income_raw.apply(lambda x: 0 if x == "<=50K" else 1)
```

Evaluating Model Performance

Student correctly calculates the benchmark score of the naive predictor for both accuracy and F1 scores.

Impressive calculation. It is always a great idea to establish a benchmark in any type of problem. As these are now considered our "dumb" classifier results, as any real model should be able to beat these scores, and if they don't we may have some model issues.

The pros and cons or application for each model is provided with reasonable justification why each model was chosen to be explored.

Please list all the references you use while listing out your pros and cons.

Very nice job mentioning some real-world application, strengths / weaknesses and reasoning for your choice! Great to know even outside of this project!

Logistic Regression

- The big thing that should be noted here is that a Logistic Regression model is a linear classifier. It cannot fit non-linear data. Thus, the model creates a single straight line boundary between the classes. (http://stats.stackexchange.com/questions/79259/how-can-i-account-for-a-nonlinear-variable-in-a-logistic-regression) (http://stats.stackexchange.com/questions/93569/why-is-logistic-regression-a-linear-classifier)
- Interpretable with some help
- Great for probabilities, since this works based on the sigmoid function

- Can set a threshold value!!

Decision Tree

- Typically very fast!
- Can handle both categorical and numerical features
- As we can definitely see here that our Decision Tree has an overfitting issue. This is typical with Decision Trees and Random Forests.
- They are are easy to visualize. Very interpretable model. Check out export_graphviz
- Another great thing that a Decision Tree model and tree methods in sklearn gives us is feature importances. Which we use later on.

Support Vector Machine

- Typically much slower, but the kernel trick makes it very nice to fit non-linear data.
- They are memory efficient, as they only have to remember the support vectors.
- Also note that the SVM output parameters are not really interpretable.
- We can use `probability = True` to calculate probabilities, however very slow as it used five fold CV

Student successfully implements a pipeline in code that will train and predict on the supervised learning algorithm given.

Very nice implementation of the `train_predict()` function!

One thing to be aware of in the future, in `fbeta_score()`, you can't switch the `y_true` and `y_pred` parameter arguments. `y_true` has to come first. Check out this example to demonstrate this

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, fbeta_score
clf = DecisionTreeClassifier(max_depth=5, random_state=1)
clf.fit(X_train, y_train)
# the correct way
print(fbeta_score(y_test, clf.predict(X_test), 0.5))
# the incorrect way
print(fbeta_score(clf.predict(X_test), y_test, 0.5))
```

Student correctly implements three supervised learning models and produces a performance visualization.

Nice work setting random states in these models and subsetting with the appropriate training set size!

Improving Results

Justification is provided for which model appears to be the best to use given computational cost, model performance, and the characteristics of the data.

Good justification for your choice in your Logistic Regression model, you have done a great job comparing these all in terms of predictive power and computational expense. Seems as your Logistic Regression model is the best of both worlds! This linear model is a pro and a con as is assume a linear boundary, but doesn't really overfit.

As in practice we always need to think about multiple things

- the predictive power of the model
- the runtime of the model and how it will scale to much more data
- the interpretability of the model
- how often we will need to run the model and/or if it support online learning.

Student is able to clearly and concisely describe how the optimal model works in layman's terms to someone who is not familiar with machine learning nor has a technical background.

Might be a bit advanced for someone who is not familiar with machine learning nor has a technical background with terms such as linear regression, sigmoid, etc....

Consider this

- Logistic Regression is a model that gives the probability of occurrence (or non-occurrence) of an event. For example, is a person likely a donor or not? This model gives a S-curve, is used to form probability of an event with given the features. The logistic regression model tries to find a decision boundary that separates the input data into two regions (donors and non-donors). Data are classified based on the probability; 0(non-donors) if less than 0.5 and 1(donors) if greater than 0.5. Once a model has been built, it can take new data and predict the probability of the person likely to be donor or non-donor.

The final model chosen is correctly tuned using grid search with at least one parameter using at least three settings. If the model does not need any parameter tuning it is explicitly stated with reasonable justification.

Great use of GridSearch here and the hyper-parameter of `C` is definitely the most tuned hyper-parameter for a LogisticRegression model. Could also tune the parameter of `penalty`, as the penalty and C parameters work hand in hand. Comparison of the sparsity (percentage of zero coefficients) of solutions when L1 and L2 penalty are used for different values of C. We can see that large values of C give more freedom to the model. Conversely, smaller values of C constrain the model more. In the L1 penalty case, this leads to sparser solutions.

Pro Tip: With an unbalanced dataset like this one, one idea to make sure the labels are evenly split between the validation sets a great idea would be to use sklearn's StratifiedShuffleSplit

```
from sklearn.model_selection import StratifiedShuffleSplit
cv = StratifiedShuffleSplit(...)
grid_obj = GridSearchCV(clf, parameters,scoring=scorer, cv=cv)
```

Student reports the accuracy and F1 score of the optimized, unoptimized, models correctly in the table provided. Student compares the final model results to previous results obtained.

Good work comparing your tuned model to the untuned one.

Pro Tip: We could also examine the final confusion matrix. A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
%matplotlib inline

pred = best_clf.predict(X_test)
sns.heatmap(confusion_matrix(y_test, pred), annot = True, fmt = '')
```

Feature Importance

Student ranks five features which they believe to be the most relevant for predicting an individual's' income. Discussion is provided for why these features were chosen.

Student correctly implements a supervised learning model that makes use of the `feature_importances_` attribute. Additionally, student discusses the differences or similarities between the features they considered relevant and the reported relevant features.

Not too shabby with your guesses above. It is tough. You may also notice that most of the 'important' features are numerical features, any ideas in why this is true? As `feature_importances_` are always a good idea to check out for tree based algorithms. If you wanted to check out the interpretability of a parameter based model, such as a Logistic Regression model, we could also check out the odd-ratio for the coefficients.

```
from sklearn.feature_selection import chi2
from sklearn.linear_model import LogisticRegression
```

```
# Fit LogisticRegression
clf = LogisticRegression().fit(X_train, y_train)
# Compute chi-squared stats between each feature and class labels
scores, pvalues = chi2(X_train, y_train)
# Create data frame
coef = pd.DataFrame.from_dict(dict(zip(clf.coef_[0].astype(np.float64),
    np.exp(clf.coef_[0].astype(np.float64)))), orient='index').reset_index()
coef.columns = ['Coef_','Odds_Ratio_']
coef['P_values'] = pvalues
coef.index = X_train.columns
# Top Five features
coef[:5]
```

How do these compare with your DecisionTreeClassifier?

Student analyzes the final model's performance when only the top 5 features are used and compares this performance to the optimized model from Question 5.

Would agree, as we always need to weigh the pros and cons here. Another idea, instead of solely picking a subset of features, would be to try out algorithms such as PCA. Which can be handy at times, as we can actually combine the most correlated/prevalent features into something more meaningful and still can reduce the size of the input. You will see this more in the next project!

Maybe something like this

```
from sklearn.decomposition import PCA

pca = PCA(n_components=0.8, whiten=True)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

clf_pca = (clone(best_clf)).fit(X_train_pca, y_train)
pca_predictions = clf_pca.predict(X_test_pca)

print("\nFinal Model trained on PCA data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, pca_pr
edictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, pca_predic
tions, beta = 0.5)))
```

DOWNLOAD PROJECT

RETURN TO PATH